VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Cloud computing na platformě Windows Azure

Cloud Computing on the Windows Azure Platform

2011 Stanislav Fajfr

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava*.

V Ostravě 29. dubna 2011 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2011 .....

Tímto bych chtěl poděkovat vedoucímu mé diplomové práce Ing. Janu Martinovičovi, Ph.D za námět, připomínky a čas, který mi věnoval.

Abstrakt

První část práce nastiňuje myšlenky cloud computingu. Diskutuje jeho strukturu, klíčové vlastnosti a typy nabízených služeb. Konec první části je věnován přehledu významných společností nabízejících cloud computing řešení. Druhá část práce podrobněji rozebírá možnosti platformy Windows Azure společnosti Microsoft a obsahuje i reálná data svědčící o skutečných vlastnostech platformy. Konec části je věnován popisu demonstrativní aplikace *CloudChat*. Třetí část se zamýšlí nad využitím výpočetního výkonu Windows Azure v oblasti *high performance computing*.

Klíčová slova: Cloud computing, Windows Azure, High performance computing, Windows HPC

Abstract

The first part contains an overview of the most significant cloud computing principles. It covers the structure, properties a various types of cloud computing services. The end of the first part lists the most popular companies together with their cloud computing offers. The second part describes the Windows Azure platform in more details and includes measured data that makes an overall picture of the real platform capabilities. At the end of the second part there is a brief introduction of the *CloudChat* example application. The final part discusses the usability of the Windows Azure platform's computing performance in a high performance computing.

Keywords: Cloud computing, Windows Azure, High performance computing, Windows HPC

Seznam použitých zkratek a symbolů

ACS	–	Access Control Service
API	–	Application Programming Interface
CLR	–	Common Language Runtime
CPU	–	Central Processing Unit
CRM	–	Customer Relationship Management
CRUD	–	Create, Read, Update, Delete
DNS	–	Domain Name System
HPC	–	High Performance Computing
HTML	–	Hypertext Markup Language
HTTP	–	Hypertext Transfer Protocol
HTTPS	–	Hypertext Transfer Protocol Secure
IaaS	–	Infrastructure as a Service
IIS	–	Internet Information Services
IM	–	Instant Messaging
IP	–	Internet Protocol
IPsec	–	Internet Protocol Security
ITIL	–	Information Technology Infrastructure Library
LAN	–	Local Area Network
LINQ	–	Language Integrated Query
MPI	–	Message Passing Interface
MSMQ	–	Microsoft Message Queuing
NAT	–	Network Address Translation
PaaS	–	Platform as a Service
PDA	–	Personal Digital Assistant
POCO	–	Plain Old CLR Object
RAM	–	Random-Access Memory
REST	–	Representational State Transfer
RPC	–	Remote Procedure Call
S+S	–	Software + Services
SaaS	–	Software as a Service
SLA	–	Service Level Agreement
SOA	–	Service-Oriented Architecture
SOAP	–	Simple Object Access Protocol

SQL	–	Structured Query Language
SSL	–	Secure Sockets Layer
TCP	–	Transmission Control Protocol
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator
VNV	–	Velmi náročný výpočet
VPN	–	Virtual Private Network
WAN	–	Wide Area Network
WCF	–	Windows Communication Foundation
WSDL	–	Web Services Description Language
XML	–	Extensible Markup Language
XMPP	–	Extensible Messaging and Presence Protocol

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Ještě před několika málo desítkami let byla většina pracovních počítačů pouhými terminály nějakého centrálního *mainframe*. Důvod je zřejmý – procesorový čas byl velmi drahý a nebylo ekonomicky únosné, aby každá pracovní stanice disponovala „větším než malým“ výpočetním výkonem.

Doba pokročila a osobní počítače se staly dostatečně výkonnými na zvládání většiny kancelářských prací. Přišla éra Internetu a z jednoduchých serverů poskytující statický obsah se staly výkonné stroje, na nichž běží sofistikované a komplexní aplikace.

V roce 2006 společnost Amazon.com, Inc. disponovala takovými serverovými výpočetními zdroji, že se rozhodla její přebytky pronajímat jiným organizacím (datová centra byla využívána na pouhých 10%). Od této chvíle je možno datovat počátek prakticky využitelného cloud computingu, kterým se ve stručnosti zabývá první část práce. Postupem času tuto cestu využití přebytečných výpočetních prostředků zvolily i další velké společnosti, jako je např. Google či Microsoft. Právě platformě Windows Azure pro cloud computing společnosti Microsoft je věnována druhá část práce.

Dnes máme možnost si pronajmout prakticky cokoliv, od hardware, software až po kompletní obchodní infrastrukturu. Cloud computing zaznamenává úspěchy i v oblasti *high performace computing*, čemž se věnuje třetí část práce.

2 Cloud computing

Jednotná definice pojmu „cloud computing“ neexistuje, protože je velice abstraktní a mediálně nadužívaný. Osobně mě zaujala definice z publikace *Cloud Computing For Dummies*:

> Cloud computing je další fází ve vývoji Internetu. Slovo *cloud* (oblak) má klíčový význam v tom, že vše – od výpočetního výkonu, infrastruktury, aplikace – může být doručeno komukoli, kdekoli a kdykoli jako ucelený produkt, resp. služba.[5, s. 8]

2.1 Vlastnosti

2.1.1 Škálovatelnost

Požadavky zákazníků jsou různorodé a mění se v čase. Jeden zákazník využije služby poskytovatele pouze několikrát ročně k testovacím účelům, kdy ale spotřebuje velký výkon. Další zákazník provozuje internetový obchod a ten se postupně rozrůstá. Zatímco obchod na počátku navštívilo několik desítek lidí denně, o pár měsíců to mohou být tisíce.

Datová centra poskytovatelů cloud computingu disponující obrovským rezervoárem výpočetních prostředků, které jsou *dynamicky* přidělovány zákazníkům. Tato dynamičnost (v literatuře je velmi často používán pojem *elasticita*) je klíčovou vlastností cloud computingu.

2.1.2 Snadná dostupnost

Služby poskytované v prostředí cloud computingu jsou obvykle velice snadno dostupné. Zřízení odběru služby či navýšení výpočetních prostředků bývá otázkou volby přepínače v softwaru spravujícím portfolio zákazníka.

2.1.3 Standardizovaná komunikace

Prakticky nutností je podpora standardizovaného komunikačního rozhraní, díky němuž je možné komunikovat mezi aplikacemi jak v rámci jednoho poskytovatele, tak i mezi různými poskytovateli. Dobrým příkladem je koncept SOA, jenž využívá všeobecně uznávaných protokolů, jako je např. WSDL, SOAP.

2.1.4 Příznivé platební podmínky

Nejrozšířenějším platebním modelem je platba pouze za to, co doopravdy spotřebujete, neboli *pay as you go*. Tento platební model má několik výhod:

- Příležitostný zákazník nemusí platit paušální poplatky za období, kdy službu nevyužívá.
- Stálý zákazník není omezen kvótami na určité období (například maximální množství procesorového času za měsíc).

Někteří poskytovatelé stále nabízejí i paušální model, ovšem i tam (většinou) platí, že pokud zákazník překročí svůj limit, budou mu dodatečně spotřebované výpočetní prostředky doúčtovány.

2.2 Typy řešení

2.2.1 Veřejný cloud

Jedná se o řešení, kde je kompletní datové centrum pod plnou kontrolou poskytovatele.

Výhody:

- Veškeré náklady na provoz a údržbu datových center nese poskytovatel.
- Velmi rychlé zavedení – zákazník pouze zažádá o danou službu a ta mu bude během chvíle doručena.

Nevýhody:

- Zákazník nemá plnou kontrolu nad svými daty. To je problém pro mnoho organizací, které nechtějí (a ani nemohou) předávat citlivá data mimo svou kontrolu.

2.2.2 Soukromý cloud

Provozovatelem datových center je samotný zákazník. Toto řešení se uplatňuje především u velkých organizací disponující značnými výpočetními prostředky.

Výhody:

- Úplná kontrola nad svými daty.
- Možnost optimalizace na podmínky organizace.

Nevýhody:

- Veškeré náklad na provoz a údržbu nese zákazník.

2.2.3 Hybridní cloud

Část cloudu tvoří datového centrum veřejného poskytovatele a část datové centrum zákazníka. Obě části bývají propojeny pomocí propojovacího subsystému, např. pomocí VPN.

Výhody:

- Data soukromého datového centra nemusí opustit hranice organizace zákazníka.
- Velkou část nákladů na provoz a údržbu nese veřejný poskytovatel (záleží na rozdělení mezi veřejnou a soukromou částí).

Nevýhody:

- Nutnost zajistit a zabezpečit komunikaci mezi soukromou a veřejnou částí.

2.3 Druhy služeb

2.3.1 Infrastructure as a Service

IaaS doručuje hardware (servery, síťové technologie, úložná zařízení a prostor v datových centrech) jako službu. Často také obsahuje operační systémy a virtualizační technologie [5, s. 107].

Zákazníci nemusí pořizovat a instalovat vlastní datové centrum. Jednoduše si předplatí výše zmíněný hardware.

Výhody dle [9, s. 35]:

- Okamžitý přístup k již nakonfigurovanému prostředí, které je většinou založeno na ITIL.
- Přístup k nejnovějším technologiím v zařízeních infrastruktury.
- Vyspělé zabezpečení a monitorování.
- Menší náklady a pracnost při rozšiřování funkcionality.

2.3.2 Platform as a Service

PaaS doručuje platformu pro aplikace zákazníků jako službu.

Jako příklad těchto služeb může posloužit společnost Google provozující Google App Engine, kde mohou zákazníci vyvíjet své vlastní aplikace v prostředí Java nebo Python a využívat funkcionalit a výkonu datových center společnosti Google.

PaaS služby umožňují zákazníkům se zaměřit na inovaci, namísto zabývání se komplexností infrastruktury [9, s. 87].

Společností nabízející PaaS je mnoho. Následuje výpis některých společných prvků [5, s. 120]:

- Platformy interně měří využívání výpočetních prostředků.
- Téměř každá platforma umožňuje zákazníkům spouštět jejich aplikace izolovaně od ostatních pomocí virtualizace.
- Platformy podporují obecně uznávané protokoly a formáty dat, jako je SOAP či XML.

2.3.3 Software as a Service

SaaS doručuje software jako službu. Poskytovatel spravuje kompletní zázemí a nese veškeré náklady na provoz, vývoj a údržbu softwaru. Zákazník si předplatí přístup k softwaru a případně možnost přizpůsobení svým potřebám.

Zdaleka nejrozšířenějším SaaS softwarem jsou aplikace typu CRM a nástroje pro správu týmové práce. Průkopníkem v této oblasti je společnost *Selaesforce.com*.

Úspěšná SaaS aplikace by měla splňovat následující [5, s. 140]:

- Musí být dostatečně obecná, tj. musí být univerzální vzhledem k různým potřebám zákazníků.
- Modulární a servisně orientovaná.
- Obsahovat nástroje pro měření a monitorování využití zákazníkem
- Zajistit izolování uživatelských dat a nastavení.
- Umožnit změnit obchodní strategii.
- Chránit uživatelská data.

Mezi výhody SaaS služeb patří [5, s. 54]:

- Automatické aktualizace SaaS aplikace ze strany poskytovatele.
- Konzistence dat v celé společnosti, tj. všichni uživatelé mají stejnou verzi software a stejná data).
- Přístup k aplikaci odkudkoli.

2.3.4 Software Plus Services

Software Plus Servise (S+S) je termín zavedený společností Microsoft a představuje variantu SaaS.

S+S aplikace nabízí většinu své funkčnosti offline. Mimo to nabízí i doplňkové služby, jejichž funkcionalitu obstarávají služby běžící v cloudu.

Mezi výhody patří především možnost práce offline. Nevýhodou jsou obvykle vyšší pořizovací náklady.

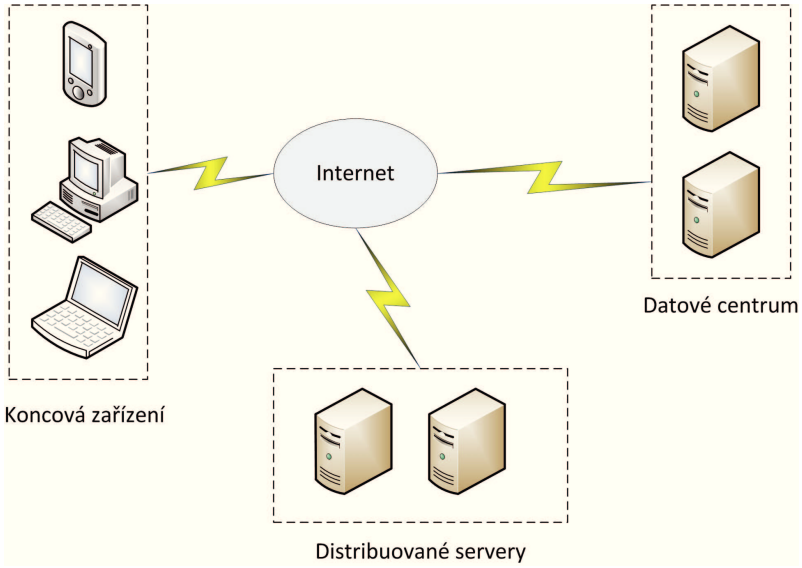
2.4 Bezpečnost

Úroveň bezpečnosti služeb hraje jednu z nejvýznamnějších rolí při výběrů poskytovatele.

Každý poskytovatel by měl jasně deklarovat:

- Kdo má (kromě zákazníka) k aplikacím a jejím datům přístup.
- Běžně se využívá vizualizačních technologií a obrazy běžících aplikací nebo celých systémů jsou duplikovány po celém světě. Jak poskytovatel zajistí smazání dat? Budou stále nějakým způsobem dostupná?
- Co se stane s daty po ukončení spolupráce mezi zákazníkem a poskytovatelem ?

Mnozí zákazníci požadující vyšší úroveň zabezpečení raději zřizují své vlastní soukromé cloudy, popř. hybridní.



Obrázek 1: Komponenty cloud computingu

2.5 Komponenty cloud computingu

Obrázek 1 zobrazuje jednotlivé komponenty tvořící koncept cloud computingu tak, jak ho popsal *Anthony Velte* v publikaci [11]. Následující text se těmito komponentami zabývá podrobněji.

2.5.1 Koncová zařízení

Koncová zařízení zprostředkovávají interakci mezi uživatelem a servery datového centra. Existuje několik variant zařízení:

Mobilní koncová zařízení – Jde o obecně mobilní zařízení, jako jsou chytré telefony či PDA.

Tenká koncová zařízení – Zařízení bez možnosti ukládat data lokálně. Aplikace běží na serveru, klient se stará pouze o zobrazení výstupu.

Tlustá koncová zařízení – Typickým představitelem je internetový prohlížeč. Častým případem jsou aplikace hostované v internetovém prohlížeči pomocí technologií Adobe Flex, Microsoft Silverlight a dalších.

2.5.2 Datové centrum

Jde o skupinu serverů a podpůrných zařízení (síťové disky, přepínače, směrovače,...), na kterých jsou provozovány předplacené aplikace, popř. které jsou zákazníkovy pronajímány.

2.5.3 Distribuované servery

Jedná se o servery (mnohdy i celá datová centra) patřící k mateřskému datovému centru. Obvykle jsou umístěny v různých lokacích po celém světě.

Mezi výhody distribuovaných řešení patří ochrana před přírodními katastrofami, možnost rozložení zátěže a redundance virtualizovaných systémů či aplikací.

2.6 Technologie

2.6.1 Komunikace

Datová centra potřebují komunikovat jak mezi sebou, tak mezi koncovými zařízeními. Vzhledem k velkému množství platforem a technologií je prakticky nutností podporovat některý z následujících transportních protokolů:

HTTP – Umožňuje spravovat[1] statické i dynamické dokumenty. Pod pojmem dokument rozumíme prakticky jakákoli data, tj. od HTML stránek až po XML fragment popisující určitý objekt.

XMPP – Jedná se o protokol založený na XML, který je známý především jako prostředek pro komunikaci IM klientů. Oproti HTTP má zásadní výhodu v možnosti obousměrné komunikace.

Webové služby komunikující nad výše zmíněnými transportními protokoly můžeme v zásadě rozdělit na dvě kategorie:

RPC webové služby – Koncept je postaven na myšlence tzv. „černých skříněk“, které jako vstup přijímají SOAP zprávu obsahující název volané operace a její argumenty. Po dokončení operace se ze skříňky vrátí SOAP zpráva obsahující výsledek operace (obrázek 2).

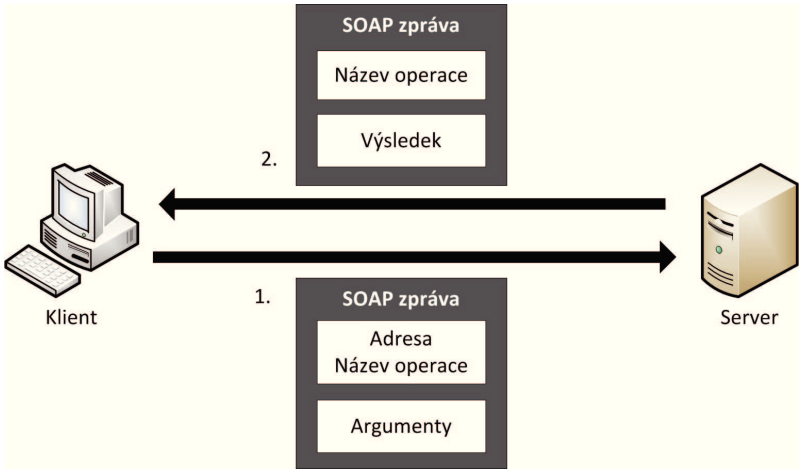
REST webové služby – REST služby nejsou o žádném speciálním protokolu, ale o architektonickém stylu [11, s. 167]. Umožňují konstruovat složité selektivní dotazy na serverová data (entity) pomocí URL, popř. tato data měnit zasíláním XML fragmentu v POST požadavku HTTP protokolu (obrázek 3). Tento styl komunikace si díky své jednoduchosti získává stále více na oblibě.

2.6.2 Zabezpečení

Zdaleka nejpoužívanějším nástrojem pro zabezpečení komunikace je protokol SSL.

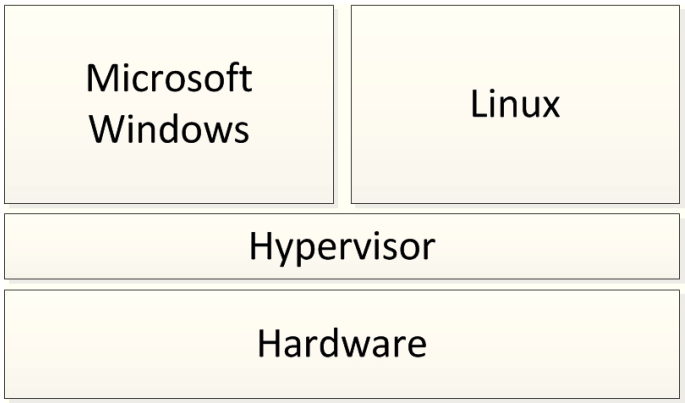
Jde o široce užívanou bezpečnostní technologii pro vytváření šifrovaných spojení mezi serverem a klientem [11, s. 157]. Je založena na SSL certifikátech, které obsahují veřejný a soukromý klíč.

[1] Základní CRUD operace jsou realizovány pomocí HTTP metod PUT, GET, POST a DELETE.



Obrázek 2: RPC webové služby

Obrázek 3: REST webové služby



Obrázek 4: Virtualizace operačních systémů

Zjednodušený postup při vytváření šifrovaného spojení:

1. Klient naváže spojení se serverem a získá jeho SSL certifikát.
2. Klient ověří platnost certifikátu.
3. Klient a server vyjednají typ šifrování.
4. Klient a server vyjednají klíč pro symetrické šifrování (do této chvíle je spojení šifrováno asymetricky).
5. Další komunikace je symetricky šifrována dohodnutým klíčem.

2.6.3 Virtualizace

Technologie virtualizace počítače, operačního systému či aplikace se uplatňuje především v prostředí cloud computingu. Bez nadsázky lze virtualizace označit za klíčový element k zajištění škálovatelnosti.

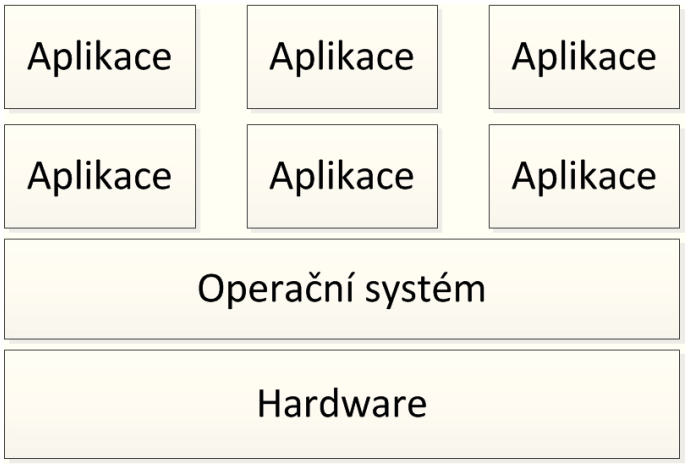
Běžným řešením je virtualizace na úrovni operačních systémů (obrázek 4), kde jednotlivé operační systémy nekomunikují přímo s hardwarem, ale s tzv. hypervisorem. Hypervisor je velmi tenké rozhraní, které zajišťuje přerozdělování a synchronizaci výpočetních prostředků hardwaru. Operační systémy obvykle nemají ani ponětí o tom, že neběží nad skutečným hardware.

Další možností je virtualizace jednotlivých aplikací (obrázek 5), kde je každá aplikace úplně izolovaná od ostatních aplikací. Veškerá komunikace s operačním systémem je virtualizována.

Virtualizace má nespočet výhod:

Možnost rozložení zátěže – Systém spravující datová centra má možnost přenášet jednotlivé virtualizované elementy[2] mezi servery a vyrovnávat tak zátěž. V opačném

[2]Elementy zde rozumímě především instance operačních systému a samostatné aplikace.



Obrázek 5: Virtualizace aplikací

případě (při malém vytížení serveru) se zbývající virtualizované elementy přenesou jinam a server je možno vypnout.

Možnost zálohování – V určitém intervalu se vytváří obraz virtualizovaného elementu. Ten je možno kdykoliv obnovit.

Podpora škálovatelnosti – Systém spravující datová centra distribuuje obraz virtualizovaného elementu na několik serverů. Na nich je element spuštěn a příchozí požadavky jsou rovnoměrně distribuovány mezi ně.

Snadná údržba – Je-li třeba určitý server odstavit, virtualizované elementy se přenesou na jiný sever.

2.7 Největší poskytovatelé cloud computingu a jejich služby

2.7.1 Amazon.com, Inc.

Společnost nabízí nespočet služeb zastřešených pod název Amazon Web Services (AWS). Následuje stručný popis nejvýznamnějších PaaS služeb.

Amazon Simple Storage Service (S3) – Slouží jako úložiště binárních datových objektů, přičemž každý objekt může být veřejný nebo soukromý. Mezi klíčové vlastnosti patří replikace a vlastní ACL.

Amazon Cloud Front – Zajišťuje distribuci dat ze služby Amazon S3. Jinými slovy jde o CDN.

Amazon Simple Queue Service (SQS) – Služba implementující vysoce škálovatelnou frontu dat.

Amazon SimpleDB – Úložiště polostrukturovaných dat. Na rozdíl od relačních databází se zde nepoužívá pevné databázové schéma. Služba mimo jiné provádí i automatickou indexaci všech dat.

Amazon Relational Database Service (RDS) – Nabízí škálovatelné instance SQL serveru MySQL.

Amazon Elastic Compute Cloud (Amazon E2) – Umožňuje provozovat obrazy operačních systémů na serverech libovolného výkonu.

2.7.2 Google

2.7.2.1 Google App Engine Jde o PaaS služby pro hostování webových aplikací. Skládá se ze tří hlavních částí:

Běhové prostředí – Aplikace v prostředí App Engine odpovídá na webové požadavky. Jakmile App Engine od klienta obdrží HTTP požadavek, identifikuje příslušnou aplikaci dle domény či subdomény. Následně vybere volný server, který požadavek zpracuje a zavolá aplikaci s přijatým HTTP požadavkem. Počká si na výsledek zpracování požadavku a ten pošle zpět klientovi. V současnosti jsou poskytována běhová prostředí pro Javu a Python.

Servery pro statické soubory – Sada severů optimalizovaných na poskytování statických souborů.

Datové úložiště – Objektová databáze. Pracuje s tzv. entitami složené z vlastností. Omezením je, že vlastnosti mohou být pouze primitivních datových typů. Samozřejmostí je podpora indexace a transakcí.

2.7.2.2 Google Docs SaaS služba zahrnující sadu webových aplikací poskytující kancelářský balík a služby pro spolupráci a plánování.

Obsahuje textový a tabulkový procesor, nástroje pro vytváření prezentací a formulářů.

2.7.3 Microsoft

Windows Azure – Sada PaaS služeb představující platformu pro webové aplikace a služby běžící v prostředí Windows Azure. Této platformě je věnována kapitola 3.

Office 365 – Kolekce kancelářských aplikací na webu a desktopu postavených nad platformou Microsoft Office. Představuje typickou SaaS službu.

3 Windows Azure

Windows Azure je cloud computing platforma umožňující běh aplikací v datových centrech společnosti Microsoft. V publikaci *Azure in Action*[3] je platformu přirovnává k operačnímu systému cloud prostředí.

Platforma poskytuje sadu služeb, které lze využívat jak v rámci Windows Azure, tak i z vnějšího světa (tj. samostatně).

Windows Azure se skládá ze tří hlavních oblastí:

Windows Azure Compute – Poskytuje výpočetní prostředky pro tzv. role.

Windows Azure Storage – Sada služeb umožňující ukládat uživatelská data v různých formách.

Windows Azure AppFabric – Rozsáhlý ekosystém služeb sloužící především k propojování aplikací.

Následující kapitoly se zabývají výše uvedenými oblastmi.

3.1 Windows Azure Compute

Při vytváření aplikací pro *Windows Azure Compute* je základní jednotkou (uživatelská) služba, která obsahuje libovolný počet rolí (obrázek 6).

Rolí se nazývá samostatná aplikace a ta může být webová nebo pracovní.

Mezi rolemi služby nemusí být žádná závislost. Jediné, co jednotlivé role služby mají společného, je definice verze operačního systému virtuálního stroje, ve kterém běží.

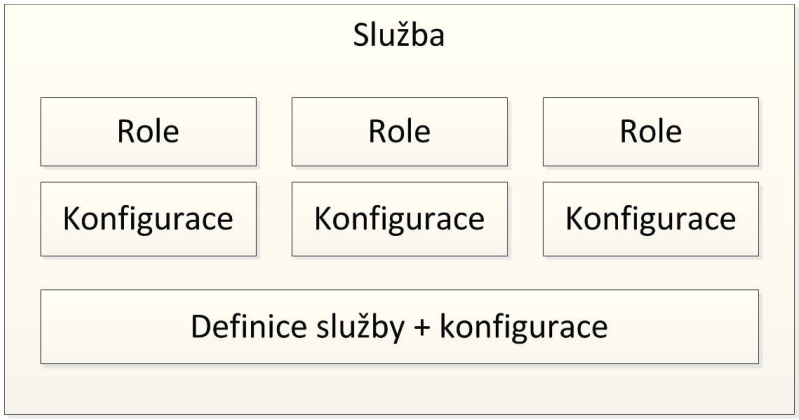
3.1.1 Role

Od každé role může být spuštěn libovolný počet instancí, mezi které je dynamicky distribuována zátěž.

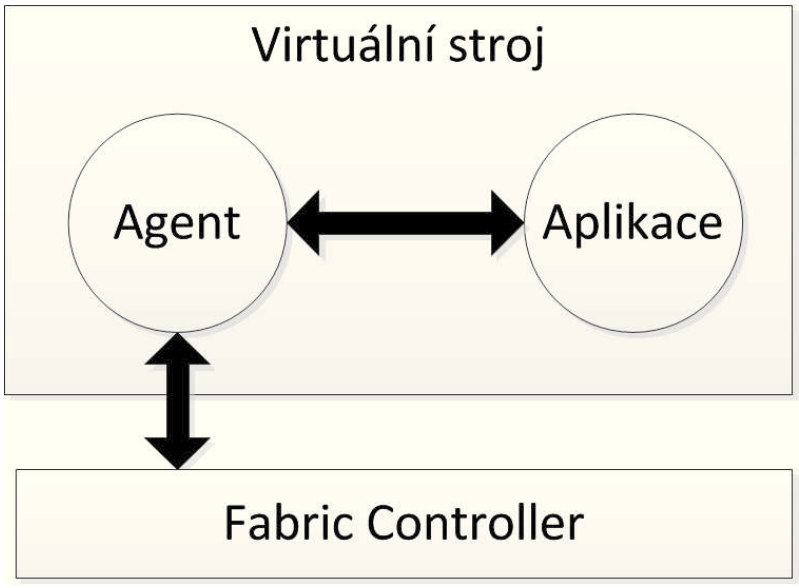
Každá instance role běží v dedikovaném virtuálním stroji. Na něm je provozován upravený operační systém Microsoft Windows Server 2008 R2. Z obrázku 7 je patrné, že role běží jako obyčejná aplikace v rámci virtuálního stroje. Proces hostující tuto aplikaci komunikuje s agentem aplikace *Fabirc Controller*.

V úvodu byla platforma Windows Azure přirovnána k operačnímu systému v prostředí cloudu. Když budeme v této analogii pokračovat, pak je aplikace *Fabric Controller* jádrem tohoto operačního systému. *Hay Chris* a *Brian Prince* ji v publikaci *Azure in Action*[3] popisuje jako „abstraktní model obrovského počtu serverů v datovém centru Windows Azure“. *Fabric Controller* má mimo jiné na starosti správu výpočetních a paměťových prostředků, správu všech služeb (vestavěných i nasazovaných zákazníky), správu rolí, zabezpečení a mnoho dalšího.

Velice důležitou součástí Windows Azure je vyvažování zátěže. *Fabric Controller* monitoruje každou instanci role a vyvažovač zátěže tak má dostatek informací ke směrování



Obrázek 6: Struktura uživatelské služby Windows Azure Compute



Obrázek 7: Windows Azure role

požadavků na nejméně zatíženou instanci. Pokud dojde k výpadku instance (např. v důsledku chyby aplikace či aktualizace operačního systému), vyvažovač zátěže vezme i tuto informaci na vědomí a zařídí se podle toho.

Počet instancí je možno konfigurovat kdykoli při běhu role. To se dá využít k automatickému navyšování a zmenšovaní počtu instancí role dle očekávané zátěže. Velice důležitou informací je, že garantovaná dostupnost 99,9% uvedená v SLA platí pouze v případě provozu minimálně dvou instancí dané role.

U každé role je možno konfigurovat velikost virtuálního stroje (tabulka 1) a možnost spouštět nativní kód. Spouštění nativního kódu je ve výchozím stavu povoleno. Pokud role nativní kód nepotřebuje, je dobré tuto volbu s ohledem na bezpečnost vypnout.

Velikost virt. stroje	CPU	RAM	Prostor na disku
Velmi malý	1 jádro (1 GHz)	768 MB	20 GB
Malý	1 jádro (1,6 GHz)	1,7 GB	250 GB
Střední	2 jádra (1,6 GHz)	3,5 GB	500 GB
Velký	4 jádra (1,6 GHz)	7 GB	1 TB
Extra velký	8 jádra (1,6 GHz)	15 GB	2 TB

Tabulka 1: Velikosti virtuálních strojů

Každá role komunikuje s aplikací *Fabric Controller*. Z toho důvodu může (pracovní role musí) obsahovat implementaci třídy RoleEntryPoint, která předepisuje několik velice důležitých metod volaných právě aplikací *Fabric Controller*:

OnStart – Tato metoda je volána při spuštění role. Zde by se měl umístit všechen inicializační kód. Role může přijímat požadavky od vyvažovače zátěže až po úspěšném skončení této metody. Metoda vrací booleovskou hodnotu indikující stav inicializace. Pokud vrací **true**, role je spuštěna a následuje volání metody Run.

Run – Metoda je volána po spuštění role. Měla by implementovat dlouhotrvající vlákno, jež se stará o práci role. Po skončení této metody se role recykluje. Výchozí implementace obsahuje uspání vlákna na nekonečnou dobu.

OnStop – Metoda je volána při ukončování role.

3.1.2 Webová role

Webová role je „klasická“ webová aplikace, přičemž je možné využít technologií ASP.NET Web Forms a ASP.NET MVC.

Oproti běžným webovým aplikacím se zde musí počítat s určitými specifiky danými prostředím cloudu. Patří mezi ně především správa informací týkající se sezení uživatele, protože v prostředí Windows Azure jsou HTTP požadavky distribuovány různým instancím rolí dle rozhodnutí vyvažovače zátěže. Zcela běžná je pak situace, kdy stejného uživatele obsluhuje několik různých instancí. Běžné webové aplikace si uchovávají informace o sezení v operační paměti serveru, v prostředí cloudu musí být tyto informace uloženy v některém ze sdílených úložišť.

Webová role přijímá HTTP požadavky od uživatelů na *externích* koncových bodech. Ty mohou být typu HTTP (na libovolném portu) nebo HTTPS (na libovolném portu s uživatelsky definovaným certifikátem).

Od každého typu smí být definován pouze jeden externí koncový bod, přičemž je povolena definice obou typů najednou (tedy HTTP i HTTPS).

Navíc lze definovat jeden *interní* koncový bod typu HTTP pro komunikaci v rámci Windows Azure (obrázek 8).

3.1.3 Pracovní role

Pracovní role je určena pro dlouhotrvající operace prováděné na pozadí. Její typický životní cyklus obvykle vypadá takto:

- Inicializace (metoda OnStart třídy RoleEntryPoint)
- Nekonečná smyčka (metoda Run třídy RoleEntryPoint)
  - Čekání a příjem požadavku
  - Zpracování požadavku
  - Odeslání/uložení výsledku
- Úklid (metoda OnStop třídy RoleEntryPoint)

Pracovní role neběží (na rozdíl od webové role) na webovém serveru IIS. Nelze s ní tedy komunikovat přímo pomocí HTTP požadavků. Pracovní role ale může hostovat WCF služby.

Pro hostování WCF služby je třeba definovat tzv. koncový bod služby. Ten může být externí nebo interní. *Externí* koncový bod přijímá požadavky od libovolných klientů a jsou směrovány pomocí vyvažovače zátěže. *Interní* koncový bod služby je určen výhradně pro komunikaci mezi rolemi v rámci Windows Azure.

Koncové body podporují protokoly HTTP, HTTPS a TCP. U externích koncových bodů je možno zvolit libovolné číslo portu. U interních koncových bodů jsou čísla portů přidělená automaticky.

Počet hostovaných WCF služeb není nijak omezen.

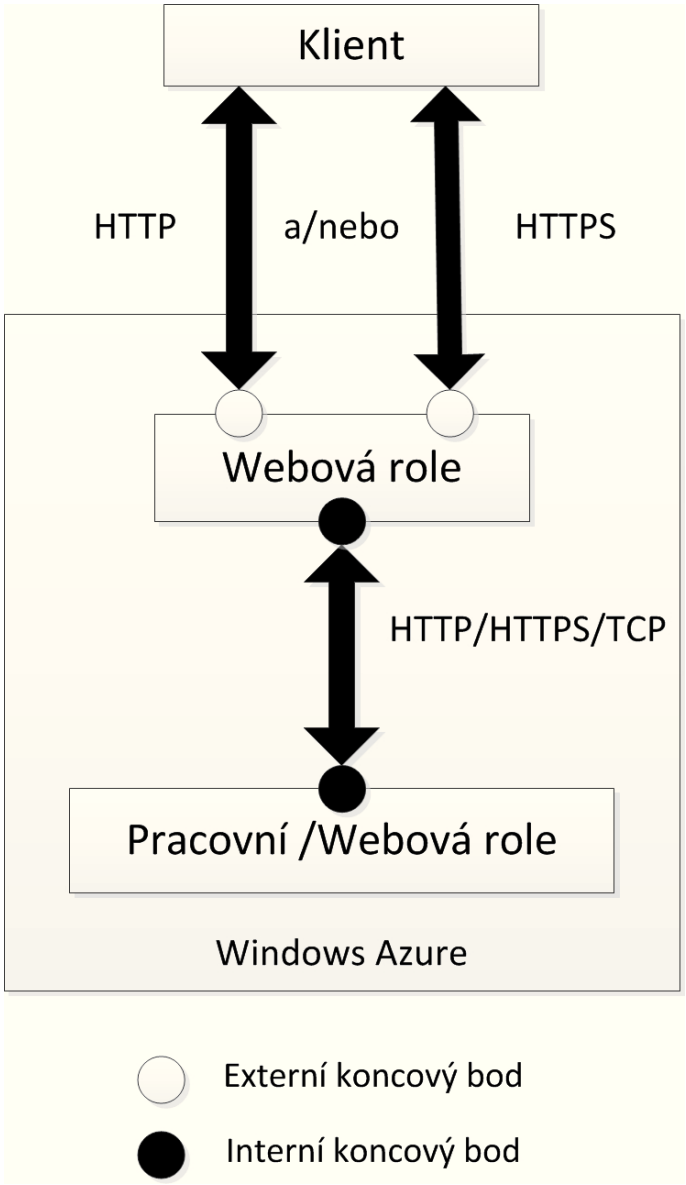
3.1.4 Reálný výkon rolí

Cílem testu výkonu rolí bylo ověřit, do jaké míry velikost role dle tabulky 1 výpovídá o reálném výkonu. Testy[3] byly provedeny na pracovních rolích všech velikostí a jako měřítko byl vybrán test *SciMark 2* portovaný na platformu .NET Framework.

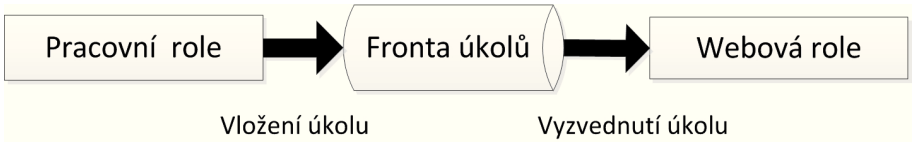
Výsledky testu jsou na první pohled překvapivé. Z měření totiž vyplynulo, že výkon všech rolí je přibližně 370 MFLOPS, tj. velikost role se na výkonu vůbec neprojevila.

Je třeba podotknout, že test *SciMark 2* čistě numerický a nepracuje s vlákny. Pokud sinyní uvědomíme, že *velikost rolí se liší v počtu jader a ne výkonu hrubém výkonu CPU*,

[3] Zdrojové kódy testovací aplikace naleznete na přiloženém CD.



Obrázek 8: Koncové body webové role



Obrázek 9: Komunikace mezi rolemi pomocí fronty zpráv

výsledek testu se stanou přijatelným. V konečném důsledku to znamená, že sílá „velkých rolí“ je v paralelizaci.

3.1.5 Komunikace mezi rolemi

Existují dvě možnosti komunikace mezi rolemi: pomocí fronty služby *Queue Service* (kapitola 3.2.3 a pomocí technologie WCF.

Fronta najde využití především v jednosměrné komunikaci. Jako příklad uveďme situaci, kdy webová role přijímá skrze webové rozhraní příkazy od uživatele. Ty jsou transformovány na zprávy obsahující popis úkolu a uloženy do fronty. Z ní si je postupně vyzvedává a zpracovává pracovní role (obrázek 9).

V případě využití WCF jedna z komunikujících rolí vystupuje jako služba a druhá jako klient. Mezi výhody patří flexibilita (např. možnost zpětného volání klienta) a možnosti zabezpečení.

3.1.6 Konfigurace služby

Konfigurace sestává ze dvou XML souborů:

ServiceDefinition.csdef – Obsahuje definici (uživatelské) služby a jejich rolí. Změna tohoto souboru vyžaduje nové nasazení, což znamená nutnost přerušení provozu všech instancí rolí v rámci služby.

ServiceConfiguration.csfg – Obsahuje upřesňující nastavení pro službu a její role. Změna tohoto souboru nevyžaduje nové nasazení.

Proč se ale zmiňovat u těchto konfiguračních souborech, když existují standardní konfigurační soubory platformy .NET Framework ? Standardní konfigurační soubory fungují samozřejmě i nadále. Je zde ovšem zásadní problém. Soubory jako web.config či app .config jsou nasazovány jako obyčejné součástí rolí. To má za následek, že pro změnu těchto souborů je třeba nové nasazení role. Z toho důvodu se doporučuje ukládat uživatelské nastavení (včetně připojovacích řetězců) do konfiguračního souboru ServiceConfiguration .csfg.

Aplikace pracuje téměř výhradně s tzv. uživatelským nastavením, což je dvojice řetězců klíč/hodnota (ekvivalent appSettings v souborech web.config a app.config). Toto nastavení musí být definováno v souboru ServiceDefinition .csdef. Zdrojový kód 1 ukazuje definici uživatelského nastavení pro připojovací řetězec DatabaseConnectionString.

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition ... >
  <WebRole name="WebRole1">
    <!-- ... -->
      <ConfigurationSettings>
          <Setting name="DatabaseConnectionString" />
    </ConfigurationSettings>
    <!-- ... -->
  </WebRole>
</ServiceDefinition>
```

Výpis 1: Definice uživatelského nastavení

Hodnota je nastavena v souboru ServiceConfiguration.csfg, jak je uvedeno ve zdrojovém kódu 2.

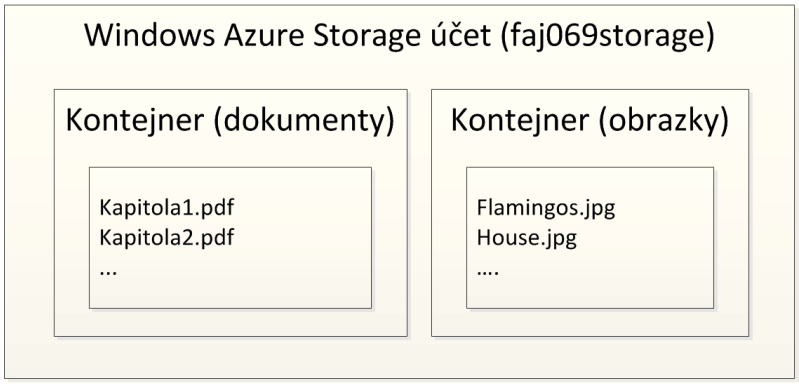
```
<?xml version="1.0"?>
<ServiceConfiguration ...>
  <Role name="WebRole1">
    <!-- ... -->
      <ConfigurationSettings>
          <Setting name="DatabaseConnectionString" value="UseDevelopmentStorage=true"
              />
      </ConfigurationSettings>
    <!-- ... -->
  </Role>
</ServiceConfiguration>
```

Výpis 2: Konfigurace uživatelského nastavení

V kódu role přistupujeme k uživatelskému nastavení pomocí třídy RoleEnvironment a metody GetConfigurationSettingValue.

Jak již bylo řečeno výše, změna uživatelského nastavení nevyžaduje nové nasazení role, natož celé služby. Proto existuje mechanismus, jakým se role dozví o těchto změnách za běhu.

```
public class WebRole : RoleEntryPoint
{
  public override bool OnStart()
  {
    RoleEnvironment.Changing += RoleEnvironmentOnChanging;
        return base.OnStart();
    }

    private void RoleEnvironmentOnChanging(object sender,
        RoleEnvironmentChangingEventArgs e)
    {
    foreach (RoleEnvironmentConfigurationSettingChange change in e.Changes.OfType<
        RoleEnvironmentConfigurationSettingChange>())
          {
            string settingName = change.ConfigurationSettingName;

            // zpracování změny
```



Obrázek 10: Struktura dat ve službě *Blob Storage*

```
                }
        }
    }
```

Výpis 3: Implementace webové role reagující na změny nastavení

Zdrojový kód 3 představuje velice jednoduchou implementaci webové role. Událost Changing třídy RoleEnvironment je volána při každé změně konfigurace role, přičemž jednotlivé změny jsou obsaženy v argumentu události. Reakce na změnu závisí na konkrétní implementaci role.

3.2 Windows Azure Storage

Windows Azure Storage zastřešuje služby poskytující různorodé přístupy k ukládání dat ve Windows Azure.

Mezi společné vlastnosti služeb *Windows Azure Storage* patří:

Škálovatelnost – Služby jsou implementovány pomocí veřejně dostupného *Windows Azure Compute* API (kapitola 3.1). To mimo jiné znamená, že každá služba běží v několika instancích a mezi ně je distribuována zátěž.

Replikace dat – Všechna data (bloby, tabulky, fronty, SQL databáze) jsou vždy replikovány na nejméně 3 bezpečnostně nezávislá místa.

3.2.1 Blob Storage

Služba *Blob Storage* pracuje s balíky binárních dat, které jsou označovaný jako „bloby“. Na obrázku 10 je znázorněna struktura dat spravovaných službou *Blob Storage*. Stejně jako u všech dalších *storage* služeb, i zde je nejvyšší jednotkou účet *Windows Azure Storage*. V něm se může nacházet libovolný počet pojmenovaných kontejnerů obsahující bloby.

Služba *Blob Storage* je založena na konceptu REST, takže je možné jakýkoli blob získat jednoduchým HTTP GET dotazem v následujícím tvaru:

http://[azurestorageucet].blob.core.windows.net/[nazevkontejneru]/[nazevblobu]

Samozřejmě je možné účet asociovat i s vlastní doménou.

Jak již bylo zmíněno, kontejner je abstraktním adresářem blobů. Kromě toho může kontejner obsahovat libovolná metadata ve formě klíč / hodnota. Existují dva druhy kontejnerů: soukromý a veřejný.

K blobům v soukromým kontejneru má přístup pouze vlastník *Azure Storage* účtu. Každý HTTP GET požadavek na některý z blobů musí být podepsán administrátorským klíčem účtu (*Account Master Key*) nebo sdíleným autentizačním klíčem.

U veřejných kontejnerů je možnost číst bloby a metadata i za pomoci anonymním HTTP požadavků. Ovšem změna dat (vložení, úprava, smazání) vyžaduje požadavek podepsaný administrátorským klíčem účtu nebo sdíleným autentizačním klíčem.

3.2.1.1 Bloby Bloby je možno rozdělit na blokové a stránkované.

Blokové bloby se skládají z bloků, které jsou identifikovatelné pomocí ID. Bloby se vytvářejí a upravují odesíláním množiny bloků a následně jejich potvrzením. Každý blok může mít maximální velikost 4 MB a velikost celého blobu nesmí přesáhnout 200 GB.

Stránkované bloby se skládají ze stránek. Stránkou je označován rozsah dat určený odsazením od začátku blobu. Při aktualizaci nebo vytváření blobu se uvádí kolekce stránek, přičemž je u každé z nich uvedeno odsazení a délka. Délky všech stránek musí být násobky 512 bytů. Maximální velikost stránkovaného blobu je 1 TB.

Ve většině případů se využívá blokových blobů, protože jsou uzpůsobeny pro proudové načítání (*streaming*) a stahování blobu jako celku. Naopak stránkované bloby se využívají v situacích, kde je potřeba častého náhodného čtení a zápisu. Typickým příkladem je obraz pevného disku virtuálního stroje.

Stejně jako kontejnery mohou bloby obsahovat metadata ve tvaru klíč/hodnota.

3.2.1.2 Zabezpečení Uvažujme příklad, kdy je k soukromým dokumentům (uložené v soukromém kontejneru) třeba povolit zápis a čtení pro vybrané obchodní partnery. Prosté rozdělení kontejnerů na soukromé a veřejné nám nedovoluje dostatečně jemné nastavení.

Služba *Blob Storage* podporuje tzv. politiku sdíleného přístupu (*Shared Access Policy*). Tato politika umožňuje na daný časový interval (např. od 1. 8. 2011 do 1. 9. 2011) povolit určité operace (čtení, procházení, zápis, mazání). Tato politika se aplikuje na kontejner a ten vygeneruje klíč sdíleného přístupu. Ten je třeba poskytnou zákazníkovi, který ho použije při vytváření HTTP GET požadavku.

3.2.1.3 Rychlost čtení a zápisu Tabulka 2 obsahuje výsledky měření[4] reálné rychlosti čtení a zápisu blokových blobů uvnitř datových center Windows Azure. S daty bylo

[4]Zdrojové kódy testovací aplikace naleznete na přiloženém CD.

manipulováno v malé webové roli, přičemž role i data byly umístěny v západní Evropě. Výsledné rychlosti jsou vážené průměry z pěti stejných pokusů.

Velikost blobu	Rychlost čtení (MB/s)	Rychlost zápisu (MB/s)
1 MB	6,8	3,8
2 MB	7,9	4,6
5 MB	9,0	5,2
7 MB	11,5	5,8
10 MB	14,6	6,5
15 MB	19,3	7,9

Tabulka 2: Naměřené rychlosti čtení a zápisu blokových blobů v rámci Windows Azure

Z výsledků nastiňující rychlosti čtení a zápisu je patrná především kolísavost. Tvůrce aplikace by měl počítat s tím, že spojení bývají nestálá.

3.2.1.4 API Jak již bylo zmíněno dříve, služba *Blob Storage* komunikuje pomocí HTTP požadavků dle koncepce REST. I když je HTTP požadavky možné sestavovat „ručně“ pomocí tříd HttpWebRequest a HttpWebResponse, lepší volbou je využití knihovny Microsoft .WindowsAzure.StorageClient.

Třída CloudStorageAccount reprezentuje *Windows Azure Storage* účet a je první komponentou, na kterou se při inicializaci obrátit. Instance se ve většině případů vytváří pomocí statické metody FromConfigurationSetting, která jako argument přijímá název uživatelského nastavení obsahující připojovací řetězec (viz. zdrojový kód 4).

Třída CloudBlobClient je proxy zaobalující komunikaci se službou *Blob Storage*. Instanci můžeme získat pomocí rozšiřující (*extension*) metody CreateCloudBlobClient třídy CloudStorageAccout nebo voláním konstruktoru, kde požadovanou adresu koncového bodu a přihlašovací údaje získáme z vlastností instance třídy CloudStorageAccount (viz. zdrojový kód 4).

Popis vybraných metod třídy CloudBlobClient:

- GetContainerReference – Vrací kontejner dle zadaného názvu.
- ListContainers – Vrací kolekci všech kontejnerů. Přetížené verze metody umožňují zadat prefix názvu požadovaných kontejnerů.

Třída CloudBlobContainer představuje kontejner blobů.

Popis vybraných vlastností třídy CloudBlobContainer:

- Name – Název kontejneru.
- Metadata – Kolekce párů klíč/hodnota. Zde je možno uchovávat libovolná doplňující data.

Popis vybraných metod třídy CloudBlobContainer:

- CreateIfNotExist – Vytvoří nový kontejner, pokud již neexistuje.
- Delete – Smaže kontejner.
- GetBlobReference – Vrací blob podle zadaného jména.
- ListBlobs – Vrací kolekci všech blobů obsažených v kontejneru.

Třída CloudBlob reprezentuje samotný blob.

Popis vybraných vlastností třídy CloudBlob:

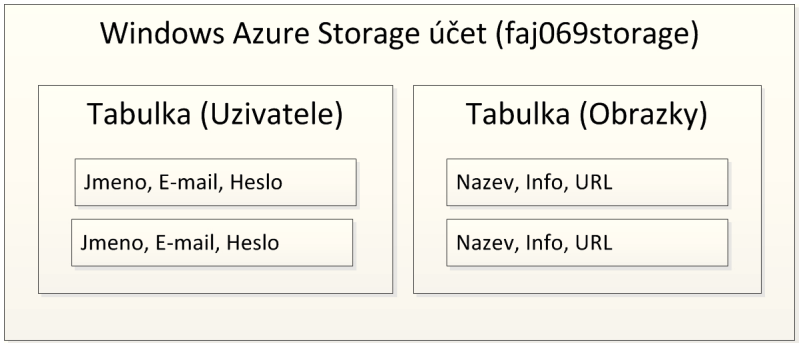
- Metadata – Kolekce párů klíč/hodnota. Zde je možno uchovávat libovolná doplňující data.

Popis vybraných metod třídy CloudBlob:

- UploadByteArray, UploadFile, UploadFromStream, UploadText – Odešle zadaná data (pole bytů, soubor, stream, text) do blobu, nad kterým je metoda volána. Pokud blob neexistuje, je vytvořen.
- DownloadByteArray, DownloadToFile, DownloadToStream, DownloadText – Stáhne kompletní obsah blobu (jako pole bytů, do souboru, do streamu, jako řetězec).
- OpenRead, OpenWrite – Vrací stream pro čtení nebo zápis. Určeno pro případ streamování obsahuje blobu.
- Delete – Smaže blob.

3.2.1.5 Příklad Zdrojový kód 4 představuje jednoduchý scénář, kdy do kontejneru *dokumenty* nahrajeme lokální soubor *Kapitola3.pdf*.

```
//získáme Azure Storage účet
CloudStorageAccount storageAccount = CloudStorageAccount.FromConfigurationSetting("
    ConnectionString");

//získáme proxy ke službě Blob Storage
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

//získáme kontejner "dokumenty"
CloudBlobContainer blobContainer = blobClient.GetContainerReference("dokumenty");

//přestože blob "Kapitola3.pdf" v kontejneru ještě neexistuje, získáme jeho referenci
CloudBlob blob = blobContainer.GetBlobReference("Kapitola3.pdf");

//je důležité zadat typ obsahue blobu, protože je tato informace důležitá pro webové prohlížeče
blob.Properties.ContentType = "application/pdf";
```



Obrázek 11: Struktura dat ve službě *Table Service*

```
//přidáme nějaká uživatelská data
blob.Metadata.Add("autor", "Stanislav Fajfr");

//nakonec odešleme obsah blobu – soubor dokumentu
blob.UploadFile(@"D:\Dokumenty\Kapitola3.pdf");
```

Výpis 4: Vytvoření nového blobu

3.2.2 Table Service

Služba *Table Service* umožňuje ukládat strukturovaná data ve formě jednoduchých tabulek (obrázek 11). Jednotlivé řádky tabulek jsou reprezentovány entitami, které mohou být na každém řádku různé, protože zde neexistuje žádné jednotné schéma.

Je velice důležité si uvědomit, že se nejedná o relační tabulky. Defacto jde pouze o „pole“ strukturovaných dat.

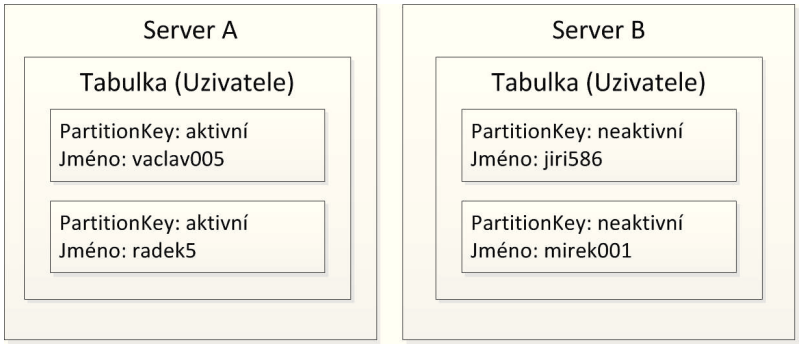
3.2.2.1 Entity Přestože se entity (řádky) v jediné tabulce mohou navzájem lišit, musí obsahovat tři povinné vlastnosti: Timestamp, PartitionKey a RowKey.

Timestamp drží datum a čas vložení či poslední aktualizace entity. Hodnota je generována na serveru a klient smí tuto vlastnost pouze číst. Jejím hlavním účelem je vypomoci při interní synchronizaci entit během souběžných požadavků. Pomocí vlastnosti PartitionKey lze jedinou tabulku rozdělit na segmenty. Hlavním účelem je možnost rozložení zátěže příliš velkých tabulek na více serverů.

Na obrázku 12 vidíme, že tabulka Uzivatele se rozdělila dle vlastnosti PartitionKey na dva segmenty: jeden obsahuje pouze aktivní uživatele a druhý pouze neaktivní. Tím dáváme službě *Table Service* najevo, že entity s vlastností hodnotou „aktivní“ vlastnosti ParitionKey mohou být uloženy na jiném serveru než entity s hodnotou „neaktivní“.

RowKey je identifikátor entity unikátní v rámci segmentu. Z toho plyne, že kombinace PartitionKey a RowKey jednoznačně identifikuje entitu v rámci celé tabulky.

Uživatelsky definované vlastnosti entity musí být primitivního datového typu. Pokud by bylo potřeba mít vlastnost ve formě entity, je možné vytvořit požadovanou vlastnost tak, aby na ni držela referenci (název tabulky, PartitionKey a RowKey).



Obrázek 12: Rozdělení tabulky Uzivatele na segmenty

Entity jsou indexovány pouze dle vlastností RowKey a PartitionKey.

Existují tzv. „zlatá pravidla“ konstrukce a dotazování entit [3, s. 219]:

- Načtení entity dle unikátní hodnoty vlastnosti PartitionKey je super rychlé.
- Načtení entity dle hodnot vlastností PartitionKey a RowKey je velmi rychlé.
- Načtení entity dle hodnoty vlastnosi PartitionKey a bez udání hodnoty vlastnosti RowKey je pomalé (je třeba přečíst všechny vlastnosti všech entit daného segmentu).
- Načtení entity bez udání hodnot vlastností PartitionKey a RowKey je velmi pomalé (je třeba přečíst všechny vlastnosti všech entity všech segmentů).

Pravidla byla potvrzena měřením[5] rychlostí vyhledávání entit, viz. graf na obrázku 13. Vyhledávání probíhalo na třech testovacích tabulkách čítajících 5000 entit, přičemž tabulky měly rozdílné rozložení řádků do segmentů.

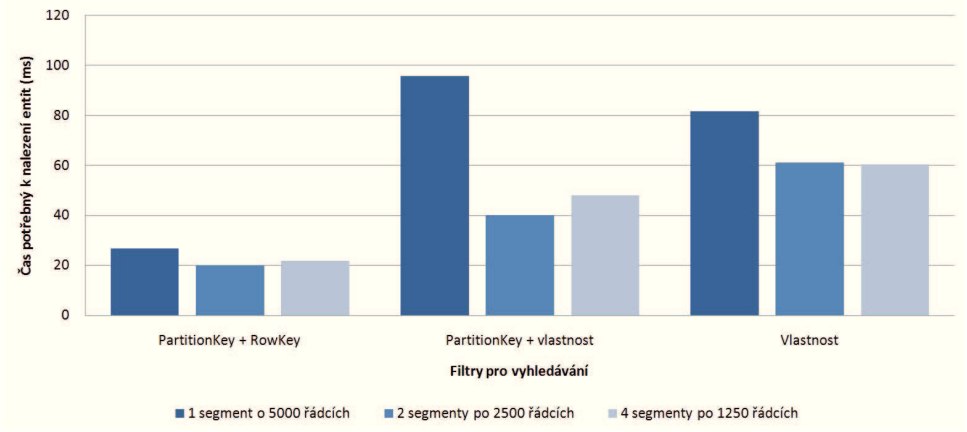
3.2.2.2 API Služba *Table Service* je postavena nad technologií *WCF Data Services*. Od této skutečnosti se odvíjí další postupy.

Nejdříve je nutno definovat entitu, přičemž existují dva způsoby. Můžeme vytvořit POCO třídu nebo implementovat třídu TableServiceEntity.

Jak bylo zmíněno výše, entita musí obsahovat tři povinné vlastnosti a libovolné množství uživatelsky definovaných. Zdrojový kód 5 definuje entitu Uzivatel jako POCO třídu.

```
[DataServiceKey("PartitionKey", "RowKey")]
public class Uzivatel
{
    public string Timestamp { get; set; }
        public string PartitionKey { get; set; }
        public string RowKey { get; set; }

        public string Jmeno { get; set; }
```

[5] Zdrojové kódy testovací aplikace naleznete na přiloženém CD.



Obrázek 13: Graf naměřených rychlostí vyhledávání v tabulce služby *Table Service*

```
        public string Email { get; set; }
        public string Heslo { get; set; }
}
```

Výpis 5: Implementace entity Uzivatel jako POCO třída

Atribut DataServiceKey sděluje infrastruktuře *WCF Data Services*, jaká kombinace vlastností entity udává unikátní klíč.

Další možností definice entity je použít abstraktní třídu TableServiceEntity, která již obsahuje zmiňované tři povinné vlastnosti. Vlastní entita je pak podtřídou této třídy (zdrojový kód 6).

```
public class Uzivatel : TableServiceEntity
{
   public string Jmeno { get; set; }
        public string Email { get; set; }
        public string Heslo { get; set; }
}
```

Výpis 6: Implementace entity Uzivatel jako podtřídu třídy TableServiceEntity

At'už je entita implementována jakkoli, vždy platí, že klient je odpovědný za korektní inicializaci všech vlastností kromě vlastnosti Timestamp.

Dalším krokem je vytvoření datového kontextu, což je implementace třídy DataServiceContext patřící to technologie *WCF Data Services*. Její odpovědností je konstrukce REST dotazů a především správa životního cyklu entit na straně klienta.

Správa životního cyklu (ve stručnosti) probíhá tak, že datový kontext zaznamenává vytváření, mazání a úpravy všech entit na straně klienta. Při volání metody SaveChanges třídy DataServiceContext se tyto záznamy projdou a vygenerují se REST požadavky, které se následně odešlou na server.

Zdrojový kód 7 udává příklad jednoduchého datového kontextu pro entity typu Uzivatel.

```
public class UzivatelDataContext : TableServiceContext
{
    public UzivatelDataContext(string baseAddress, StorageCredentials credentials)
        : base(baseAddress, credentials)
    {
    }

    public IQueryable<Uzivatel> Uzivatele
    {
        get
        {
            return this.CreateQuery<Uzivatel>("Uzivatele");
        }
    }
}
```

Výpis 7: Implementace datového kontextu pro entitu Uzivatel

Je samozřejmě možné vytvořit další pomocné metody pro přidání a smazání entity, které budou v konečném důsledku volat metody nadtřídy AddObject či DeleteObject a následně SaveChanges.

3.2.2.3 Získávání dat Vlastnost Uzivatele datového kontextu UzivateleDataContext ze zdrojového kódu 7 vrací rozhraní IQueryable<Uzivatel> známé z technologie LINQ. Tím nám datový kontext říká, že LINQ konstrukce vnitřně transformuje na odpovídající REST požadavky.

Bohužel technologie *WCF Data Services* podporuje jen velmi malou podmnožinu LINQ metod. Podporované jsou pouze metody Where, Take, First a FirstOrDefault. Chybí také podpora pro vnořené dotazy a některé specifické konstrukce.

3.2.2.4 Administrace tabulek Datový kontext technologie *WCF Data Services* neposkytuje žádné API pro administraci tabulek. Od toho je zde třída CloudTableClient.

Popis vybraných metod třídy CloudTableClient:

- CreateTable – Vytvoří novou tabulku se zadaným názvem.
- DeleteTable – Smaže tabulku dle zadaného názvu.
- ListTables – Vrací názvy všech tabulek v rámci účtu *Windows Azure Storage*.

3.2.3 Queue Service

Třetím členem rodiny *storage* služeb je služba *Queue Service*, neboli fronta zpráv. Jedná se o FIFO kolekci zpráv, přičemž jejich počet není omezen. Každá fronta je identifikovatelná svým názvem a může obsahovat libovolná metadata (maximální velikosti 8 KB).

Prakticky vše je zde podřízeno rychlosti. Z toho plyne koncepce zpráv – mnoho malých bloků dat. Zpráva může obsahovat pouze řetězec nebo pole bytů a její maximální velikost

je 8 KB. Pokud bychom ke zprávě chtěli připojit větší množství dat, můžeme postupovat tak, že si velká data uložíme např. do *Blob Storage* a do fronty vložíme URL blobu.

Jedním z cílů služby je, aby každá zpráva byla zpracována pouze jednou. To má za následek, že zprávy nejsou při vyzvednutí implicitně smazány, protože mazání je odpovědnost příjemce. Může ale nastat případ, kdy se příjemce zprávy dostane během jejího zpracování do potíží. Jak tuto situaci ošetřit?

Proces vyzvednutí zprávy vypadá následovně:

1. Příjemce překontroluje, zdali fronta obsahuje nějaké zprávy.
2. Příjemce vyzvedne zprávu a při tom zadá *visibility timeout*. Jde o časový limit, do kterého bude zpráva pro všechny ostatní příjemce neviditelná.
3. Příjemce zpracuje zprávu a smaže ji z fronty. Pokud to vše nestihne do časového limitu zadaného jako *visibility timeout*, zpráva se stane opět viditelnou pro ostatní příjemce.

Z výše popsaného postupu plyne důležitost správné volby hodnoty *visibility timeout*. Pokud bude příjemci trvat zpracování o něco déle, než je hodnota *visibility timeout*, zprávu začne zpracovávat jiný příjemce. Obecným doporučením je provádět zpracování zprávy tak, aby i nechtěné opakované zpracování nemělo negativní dopad na stav systému.

3.2.3.1 API Hlavním účelem tříy CloudQueueClient je získávání referencí na zadané fronty. Instanci můžeme jednoduše získat pomocí *extension* metody CreateCloudQueueClient třídy CloudStorageAccout nebo voláním konstruktoru, kde požadovanou adresu koncového bodu a přihlašovací údaje získáme z vlastností instance třídy CloudStorageAccount (viz. zdrojový kód 8).

Popis vybraných metod třídy CloudQueueClient:

- GetQueueReference – Vrací frontu dle jména.
- ListQueues – Vrací názvy všech front v rámci účtu *Windows Azure Storage*.

Třída CloudQueue představuje frontu zpráv.

Popis vybraných vlastností třídy CloudQueue:

- ApproximateMessageCount – Vrací přibližný počet zpráv ve frontě. Tato informace je průběžně aktualizována, takže se může stát, že v aktuální hodnotě nejsou započteny nově přidané nebo smazané zprávy.
- Metadata – Kolekce párů klíč/hodnota. Zde je možno uchovávat libovolná doplňující data.

Popis vybraných metod třídy CloudQueue:

- AddMessage – Přidá novou zprávu na konec fronty.
- DeleteMessage – Smaže zadanou zprávu.
- GetMessage – Vrací zprávu ze začátku fronty a nastavuje *visibility timeout* (výchozí hodnota je 30 sekund).
- PeekMessage – Vrací zadanou zprávu a nenastavuje *visibility timeout*.
- DeleteMessage – Smaže zadanou zprávu.

Třída CloudQueueMessage reprezentuje zprávu ve frontě.

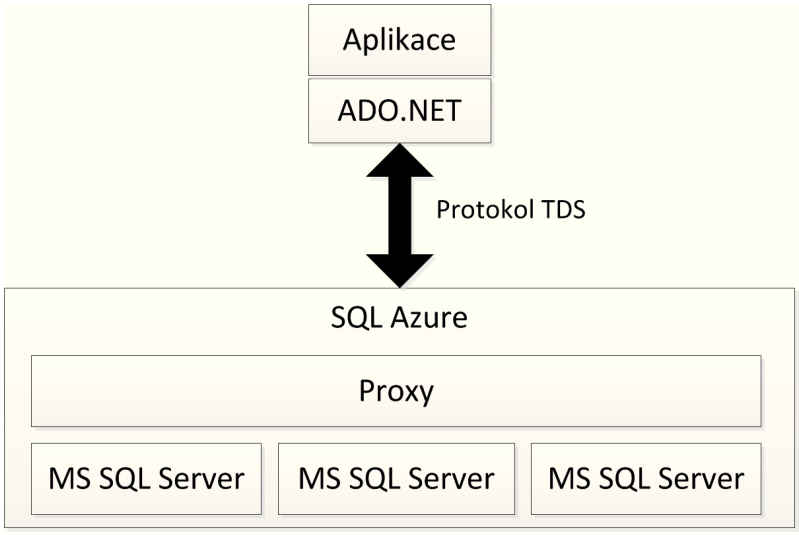
Popis vybraných vlastností třídy CloudQueueMessage:

- AsBytes – Vrací obsah zprávy jako pole bytů.
- AsString – Vrací obsah zprávy jako řetězec.
- InsertionTime – Vrací datum a čas vložení zprávy do fronty.
- NextVisibleTime – Vrací datum a čas, kdy bude zpráva opět viditelná pro ostatní příjemce.

3.2.3.2 Příklad Zdrojový kód 8 představuje jednoduchý scénář, kdy do fronty prace vložíme novou zprávu s obsahem „1+1“ a následně ji vyzvedneme s hodnotou *visibility timeout* nastavenou na jednu minutu.

```
//získáme Azure Storage účet
CloudStorageAccount storageAccount = CloudStorageAccount.FromConfigurationSetting("
    ConnectionString");

//získáme proxy ke službě Queue Sevice
CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

//získáme frontu "prace"
CloudQueue queue = queueClient.GetQueueReference("prace");

//vytvoříme zprávu s obsahem "1+1" a vložíme ji do fronty
CloudQueueMessage queueMessage = new CloudQueueMessage("1+1");
queue.AddMessage(queueMessage);

//  ...

//z  fronty  vyzvedneme zprávu a nastavíme její neviditelnost pro ostatní
//příjemce na jednu minutu
CloudQueueMessage message = queue.GetMessage(TimeSpan.FromMinutes(1));
```

Výpis 8: Vložení a vyzvednutí zprávy do/z fronty



Obrázek 14: Architektura služby *SQL Azure*

3.2.4 SQL Azure

Téměř každá aplikace využívá k ukládání svých dat relační databáze. Přestože zde existuje služba *Table Service* (kapitola 3.2.2), rozhodně není jejím účelem nahradit relační SQL databázi. Z toho důvodu je součástí *storage* služeb služba *SQL Azure*. Jde o skutečnou SQL databázi založenou na systému Microsoft SQL Server 2008 R2.

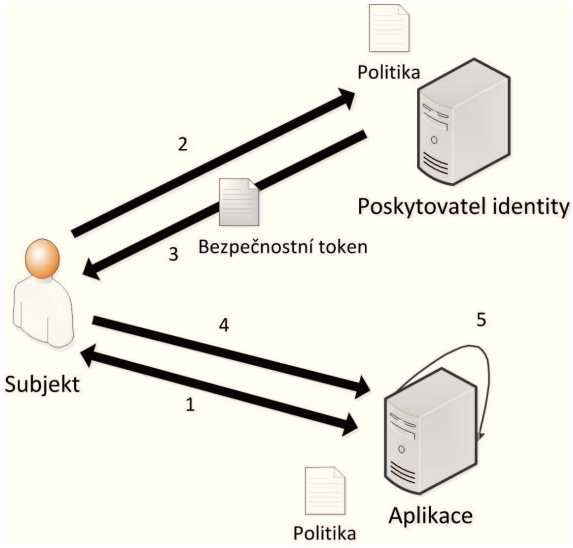
Je zde ovšem několik omezení, která souvisí se speciální implementací SQL databáze v prostředí cloudu. Nepodporuje integraci CLR, zrcadlení databáze, distibuované dotazy a transakce, správu *filegroup* a fulltextové vyhledávání.

Klientské aplikace mohou se službou *SQL Azure* pracovat úplně stejným způsobem a stejnými nástroji jako s jinými SQL databázemi. Ve vývojářích to může vytvářet dojem, že se připojuje ke skutečné SQL databázi. Obrázek 14 okazuje, že tomu tak (úplně) není.

Klientská aplikace komunikuje s databází pomocí standardního subsystému ADO.NET. Ten komunikuje se službou *SQL Azure* pomocí protokolu TDS. Požadavky však nedostávají skutečné SQL Servery, ale proxy služba „tvářící se“ jako SQL Server. Ta má za úkol požadavky analyzovat (kvůli bezpečnosti, optimalizaci) a následně je předat některé instanci SQL Serveru. Z důvodu existence proxy služby vznikají zásadní omezení. Není možné použít žádný SQL příkaz vztahující se k fyzickým aspektům infrastruktury.

3.2.4.1 Replikace dat Replikace databází je zde poněkud komplikovanější než v případě ostatních *storage* služeb.

V každém okamžiku existují minimálně tři repliky databáze. Jedna z nich je vždy primární. Právě na primární repliku se posílají SQL dotazy, přičemž její stav se automaticky replikuje na ostatní databáze čekající na svou příležitost. Při nedostupnosti primární repliky je za novou primární repliku určena jedna ze záložních.



Obrázek 15: Schéma zabezbečení založeném na tvrzeních

3.3 Windows Azure AppFabric a jiné služby

3.3.1 Access Control Service

Access Control Service (dále jen ACS) poskytuje klientským aplikacím možnost externalizovat správu a autentizací uživatelů. Služba je založena na bezpečnostním modelu pracujícím s tvrzeními (*claims*).

Zabezpečení založené na tvrzeních funguje na spolupráci několika entit, z nichž každá má svou přesně stanovenou roli.

Subjekt – Subjekt je cokoli (uživatel, aplikace), co se snaží získat přístup do požadované aplikace.

Poskytovatel identity – Služba spravující identity subjektů. Poskytuje také autentizaci uživatelů.

Aplikace – Jde o aplikaci, k níž se snaží subjekt získat přístup a která důvěřuje definovaným poskytovatelům identit. Aby subjekt získal přístup, musí předložit bezpečnostní token obsahující kolekci tvrzení o subjektu. Bezpečnostní token musí pocházet od důvěryhodného poskytovatele identity a nesmí být cestou změněn.

Bezpečnostní token – Objekt obsahující množinu tvrzení u subjektu. Token je vydán příslušným poskytovatelem identity a je jím digitálně podepsán.

Obecné schéma zabezpečení (viz. obrázek 15) vypadá následovně [2]:

1. Subjekt chce získat přístup k aplikaci obvykle pomocí nějakého agenta (např. webový prohlížeč). Subjekt si přečte politiku aplikace, která obsahuje seznam důvěryhodných poskytovatelů identity, požadovanou sadu tvrzení (např. jméno, datum narození) a podporované bezpečnostní protokoly.
2. Subjekt si vybere jednoho z podporovaných poskytovatelů identity a provede inspekci jeho politiky, kde zjistí, jaký má použít bezpečnostní protokol a jaké údaje má předložit ke své identifikaci. Následně mu zašle požadavek (společně s požadovanými přihlašovacími údaji) na vydání bezpečnostního tokenu.
3. Poskytovatel identity ověří požadavek subjektu a v případě úspěchu vydá bezpečnostní token obsahující tvrzení požadovaná aplikací.
4. Subjekt obdrží bezpečnostní token od poskytovatele a zašle ho spolu s prvním požadavkem aplikaci.
5. Aplikace ověří příchozí bezpečnostní token dle své politiky a v případě úspěchu subjektu povolí přístup.

ACS nabízí služby poskytovatele identity, ovšem v mírně odlišné a užitečnější formě. ACS se aplikacím jeví jako obyčejný poskytovatel identity. Podporuje několik poskytovatelů identity třetích stran (Google, Yahoo a další). Oproti nim vystupuje jako aplikace – to znamená, že tvrzením vydaných těmito poskytovateli důvěřuje. Administrátor ACS při konfiguraci svých aplikací mimo jiné nastaví, které poskytovatele identity třetích stran má ACS podporovat.

Subjekty (uživatelé) přistupující k aplikaci jsou přesměrováni na stránku ACS, kde si vyberou poskytovatele identity, u které mají účet (Windows Live, Google). Následně jsou přesměrování na přihlašovací stránku vybraného poskytovatele a po úspěšné autentizaci se dostanou zpět do aplikace.

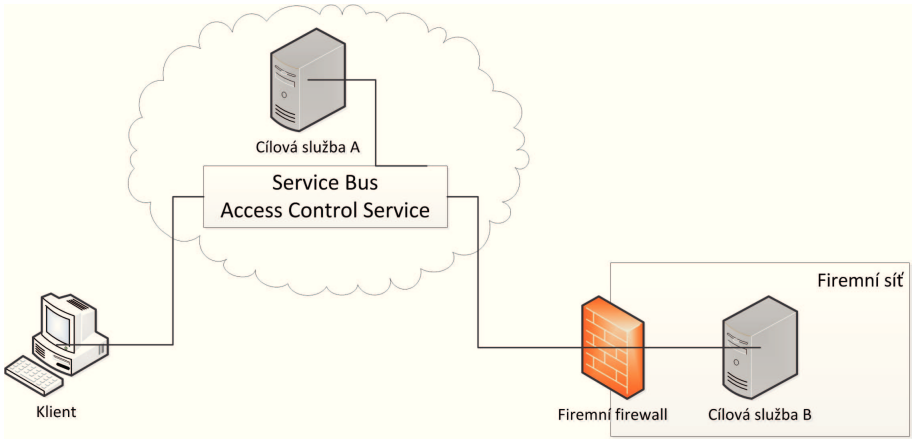
Nezáleží na tom, u kterého poskytovatele identity se subjekt autentizuje. Aplikace důvěřuje ACS a když budou tvrzení v bezpečnostním tokenu podepsána ACS, pak aplikace subjektu povolí přístup.

3.3.2 Service Bus

Mezi nejvýznamnější služby Windows Azure patří sběrnice služeb *Service Bus*. Jde o sadu služeb zajišťující konektivitu podporující co nejmenší vazbu mezi cílovými službami a jejich klienty. Roli sběrnice služeb ukazuje obrázek 16.

Stejně jako většina služeb Windows Azure je i celá sběrnice postavena na technologii WCF. Právě proto je API sběrnice postavené na rozšíření WCF.

Spojová služba (*relay service*) řeší problémy konektivity mezi cílovými službami a jejich klienty v síti Internet. Mnohé cílové služby jsou ukryty ve firemních sítích za různými síťovými prvky (firewally , vyvažovače zátěže, NAT). Provozovatelé těchto služeb nejsou nadšeni nutností jejich „násilnému“ vystavení na Internet pomocí statických IP adres, demilitarizovaných zón apod. Navíc mnoho služeb by rádo uvítalo možnost zpětného volání klientovi (tj. duplexní spojení).



Obrázek 16: Role sběrnice služeb ve Windows Azure

> Pokud je nemožné vytvořit přímé spojení mezi klientem a cílovou službou, řešením je mezi ně vložit neutrálního prostředníka – spojovou službu. Její zodpovědností je zprostředkovávat spojení a předávat zprávy. [6]

Předpokladem je, že všichni účastníci komunikace mají možnost vytvoření odchozího spojení ke spojové službě.Zjednodušený scénář zprostředkování konektivity pomocí spojové služby:

1. Nejdříve se musí cílová služba autentizovat u spojové služby. Pokud žádosti vyhoví, spojová služba začne naslouchat na zadaném koncovém bodu sběrnice. Tento bod se od této chvíle stává zástupcem cílové služby.
2. Klient se autentizuje u spojové služby.
3. Klient zašle zprávu pro cílovou službu přes spojovou službu.
4. Cílová služba obdrží zprávu od klienta skrze spojovou službu.

Z pohledu klienta i cílové služby není práce se spojovou službou o moc jiná než s obyčejnými WCF službami a klienty. Sběrnice plně podporuje spolehlivé spojení a zabezpečení na úrovni zpráv i transportu. Naopak chybí podpora distribuovaných transakcí.

Sběrnice podporuje několik modelů komunikace. Ty se nastavují použitím speciálních *binding*[6] sběrnice:

netTcpRelayBinding – Podporuje modely požadavek / odpověď, jednosměrný a duplexní. Je možno nastavit tři metody spojení:

Zprostředkovaně – Výchozí metoda, kdy veškerá komunikace probíhá vždy přes spojovou službu.

[6]Jedná se o termín z technologie WCF popisující technologii propojení mezi službou a klientem.

Přímo – Inicializace spojení probíhá skrze spojovou službu, která následně klientovi sdělí přímou adresu cílové služby a vytvoří se mezi nimi přímé spojení.

Hybridně – Inicializace probíhá opět skrze spojovou službu. Následně se ověří, zdali je cílová služba dosažitelná klientovi. Pokud ano, vytvoří se přímé spojení. Pokud ne, ponechá se zprostředkované.

netOnewayRelayBinding – Obsahuje pouze jednosměrný model komunikace. Klient zašle zprávu spojové službě, kde se uloží do fronty (na rozdíl od *Queue Service* zde není zaručeno pořadí a maximální doba uložení je přibližně 10 minut). Odtud si ho cílová služba vyzvedne. To mimo jiné znamená, že cílová služba nemusí být v době odesílání zprávy od klienta připojena ke sběrnici. Na druhou stranu zde není žádná možnost zaslání odpovědi zpět klientovi.

netEventRelayBinding – Jde o specializaci netOnewayRelayBinding, která obsahuje možnost naslouchání více cílových služeb na jednom koncovém bodě sběrnice. Jde tedy o primitivní způsob broadcastu zpráv od klientů směrem k naslouchajícím koncovým službám. Spojová služba zde figuruje v roli směšovače (hub).

Výše byly vyjmenovány nejzajímavější *binding*. Téměř každý standardní WCF *binding* má speciální „relay" verzi pro použití u sběrnice (např. basicHttpRelayBinding, webHttpRelayBinding).

3.3.2.1 Zabezpečení Jak je popsáno v předchozí kapitole, cílová služba i klient se musí u sběrnice autentizovat, přičemž autentizaci má na starosti služba *Access Control Service* (kapitola 3.3.1). Existují celkem 4 způsoby autentizace:

Žádná – Pouze na straně klienta. Cílová služba se musí autentizovat vždy.

Sdílené tajemství – Řešení podobné obvyklé dvojici – jménu a heslu. Tajemství je hodnota nastavená v administraci sběrnice.

SAML 2.0

SWT pomocí protokolu WRAP

Cílová služba má při připojování ke sběrnici možnost potlačit nutnost autentizace klientů.

Vzhledem k tomu, že sběrnice je postavena na technologii WCF, nic nebrání využití standardních bezpečnostních prvků WCF, jak je např. zabezpečení na úrovni zpráv.

3.3.2.2 Adresování a registry služeb Každá služba nasazovaná v rámci sběrnice musí patřit k nějakému řešení, což je virtuální prostor založený v administraci, který má přidělen jedinečné DNS jméno v rámci celé sběrnice.

Kompletní adresa služby má schéma:

[protocol]://[název řešení]. servicebus.windows.net/[název služby]

Kde:

- Protokol může být buď hodnota „sb“ (*service bus*) nebo „http“ ,popř. „https“.
- Název služby může být jakýkoli platný DNS záznam.

Sběrnice služeb poskytuje na každé základní adrese řešení a na všech jejich podadresách ATOM kanál publikující seznam veřejných služeb. ATOM kanál obsahuje všechny veřejné služby obsažené v podadresách, tzn. že respektuje adresářovou hierarchii. [7]

Příklad: mějme vystavené tři veřejné služby na následujících adresách:

- *sb://faj069test.servicebus.windows.net/tools/echoA*
- *sb://faj069test.servicebus.windows.net/tools/echoB*
- *sb://faj069test.servicebus.windows.net/main*

ATOM kanál na adrese *https://faj069test.servicebus.windows.net/tools/* bude obsahovat adresy služeb *echoA* a *echoB*. ATOM kanál na adrese *https://faj069test.servicebus.windows.net/* bude obsahovat adresy sližeb *main*, *echoA* a *echoB*.

3.3.3 Windows Azure Connect

Tato služba slouží k propojení počítačů v síti zákazníka s instancemi Windows Azure rolí na úrovni protokolu IP (skrze IPsec). Obrázek 17 ukazuje příklad propojení.

Windows Azure Connect vytváří logicku virtuální síť, která může obsahovat dva typy skupin: skupinu rolí a skupinu počítačů.

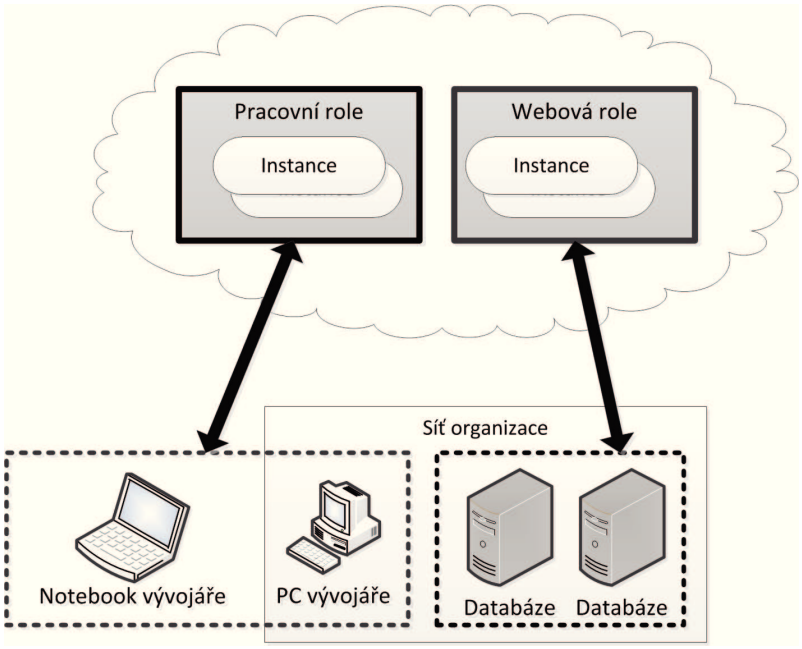
Skupina rolí se skládá z předem definovaných rolí. Jednotlivé instance rolí jsou automaticky přidávány a odebírány do/ze skupiny. Skupina počítačů se skládá z počítačů vně Windows Azure, které mají nainstalovanou a nakonfigurovanou aplikaci *Windows Azure Connect Endpoint Software*. Počítač může být vždy pouze v jediné skupině.

Jak plyne z obrázku 17, skupina rolí může být spojena se skupinou počítačů (dokonce s několika najednou). Určitou limitací je, že členové skupiny rolí nemají možnost spojení mezi sebou. Skupiny počítačů mohou být (kromě skupin rolí) připojeny i k jiným skupinám počítačů.

Každá skupina počítačů má možnost povolení vzájemné komunikace. Díky tomu lze např. připojit domácí počítač vývojáře do firemní sítě, i když je domácí počítač ukryt za NAT (tj. není veřejně přístupný).

Mezi další možnosti patří:

- Připojení instance Windows Azure role do domény technologie Active Directory.
- Vzdálená administrace a ladění instancí Windows Azure role.
- Správa instancí Windows Azure role pomocí stávajícího softwaru ve firemní síti, např. pomocí Windows PowerShell.



Obrázek 17: Propojení počítačů s Windows Azure rolemi pomocí služby *Windows Azure Connect*

3.4 Náklady na provoz Windows Azure aplikací

Cílem této kapitoly je podat rámcový přehled nákladů na provoz aplikací ve Windows Azure. Zároveň jsou pro porovnání uvedeny náklady *on-premise* [7].

Poznámky úvodem:

- Scénáře a požadavky jsou pouze ilustrační.
- Ceny jsou uvedeny bez DPH a jsou platné k datu 21. 3. 2011.
- Pro nabídky na on-premise řešení byla aplikována tato omezení:
  - Provozovatel musí působit v České republice.
  - Musí být k dispozici pronájem licencí společnosti Microsoft.

3.4.1 Malý webový server

Požadavky: 1 CPU, 2 GB RAM, Microsoft Windows Server 2008, SQL databáze o velikosti 1 GB

Windows Azure nabídka:

[7] Výrazem on-premise je zde myšleno jakékoli řešení mimo cloud computing.

- 1 malý virtuální stroj (1x CPU 1,6 GHz, 1,75 GB RAM)
- SQL Azure velikosti 1 GB
- Cena za měsíc: 1703,00 Kč

On-premise nabídka (wedos.cz):

- Virtuální server (2 jádra, 2 GB RAM)
- Pronájem licence na Microsoft Windows Server 2008 R2 Datacenter
- Microsoft SQL Server 2008 R2 Express (max. 4 GB)
- Cena za měsíc: 600 Kč

3.4.2 Středně velký webový server

Požadavky: 2 CPU nebo jádra, 4 GB RAM, Microsoft Windows Server 2008, SQL databáze o velikosti 5 GB

Windows Azure nabídka:

- 1 středně velký virtuální stroj (2x CPU jádra 1,6 GHz, 3,5 GB RAM, 60 GB diskového prostoru)
- SQL Azure velikosti 5 GB
- Cena za měsíc: 3 932,00 Kč

On-premise nabídka (ignum.cz):

- Dedikovaný server (Basic FULL - Intel Xenon Quad Core, 4 GB RAM, 2x300 GB diskového prostoru)
- Pronájem licence Microsoft Windows Web Server 2008 R2
- Pronájem licence Microsoft SQL Server 2008 Web Edition
- Cena za měsíc: 4 765,00 Kč

3.4.3 Velký webový server

Požadavky: 4 CPU nebo jádra, 8 GB RAM, Microsoft Windows Server 2008, SQL databáze o velikosti 10 GB

Windows Azure nabídka:

- 1 velký virtuální stroj (4x CPU jádra 1,6 GHz, 7 GB RAM, 1 TB diskového prostoru)

- SQL Azure velikosti 10 GB
- Cena za měsíc: 7 882,00 Kč

On-premise nabídka (vhosting.cz):

- Dedikovaný server (Intel Xenon 8 jader, 8 GB RAM, 2x500 GB diskového prostoru)
- 200 GB záloha na 7 dní
- Pronájem licence Microsoft Windows Web Server 2008 R2
- Pronájem licence Microsoft SQL Server 2008 Web Edition
- Cena za měsíc: 5 742,00 Kč

3.4.4 Vyhodnocení

Tabulka 3 udává celkový výsledek srovnání nákladů (za měsíc) výše uvedených scénářů. Náklady na provoz aplikací ve Windows Azure jsou poměrně vysoké. Ze srovnání vychází, že Windows Azure je optimální pro provoz středně velkých aplikací. Nejnákladnější z nabízených služeb Windows Azure je služba *SQL Azure*. Naopak velice výhodné jsou ceny služeb *Azure Storage*.

	Malý web. server	Středně velký web. server	Velký web. server
Windows Azure	1 703,00 Kč	3 932,00 Kč	7 882,00 Kč
On-premise	600,00 Kč	4 765,00 Kč	5 742,00 Kč

Tabulka 3: Porovnání nákladů mezi Windows Azure a *on-premise* řešeními

3.5 Příklad: aplikace CloudChat

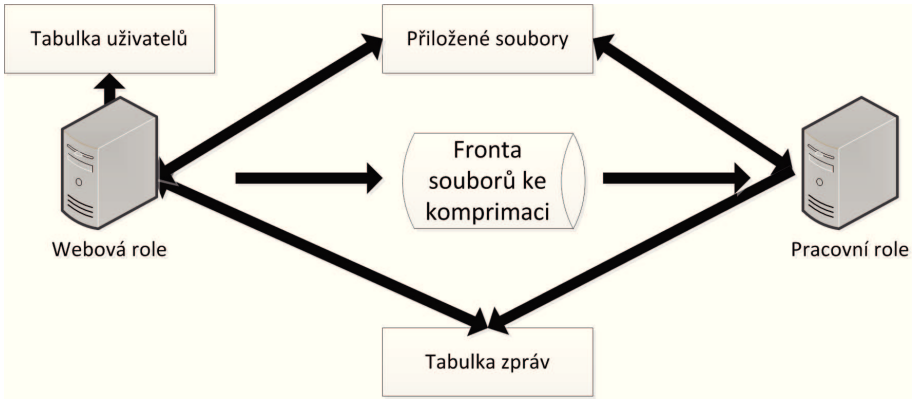
CloudChat je jednoduchá webová aplikace (zdrojové kódy naleznete na přiloženém CD) umožňující vkládat krátké zprávy do globálního úložiště zpráv, ke kterému mají přístup všichni návštěvníci.

Uživatelé se neregistrují. Na úvodní stránce si zvolí uživatelské jméno, které musí být v danou chvíli volné (tj. nikdo s daným jménem nesmí být v chatu přítomen). Poté je přesměrován na hlavní stránku obsahující seznam zpráv a formulář pro vložení nové zprávy.

Uživatel může ke své zprávě přiložit libovolný soubor. Navíc může aplikaci instruovat, aby byl přiložený soubor zkomprimován (tj. vytvořil se „.zip soubor“).

CloudChat zahrnuje i administrativní stránku, kde je možné spravovat uživatele, zprávy a přiložené soubory.

Strukturu aplikace zachycuje obrázek 18. Webové stránky aplikace *CloudChat* běží ve webové roli. Komprimaci souborů provádí pracovní role.



Obrázek 18: Struktura aplikace *CloudChat*

3.5.1 Entity

Seznamy aktivních uživatelů je uchováván v tabulce služby *Table Service* (kapitola 3.2.2) jako kolekce entit typu ChatUser (obrázek 19). Eviduje se pouze uživatelské jméno, které je s výhodou ukládáno přímo jako klíč RowKey. Díky tomu je častá kontrola existence uživatelského jména velice rychlá (viz. kapitola 3.2.2.1).

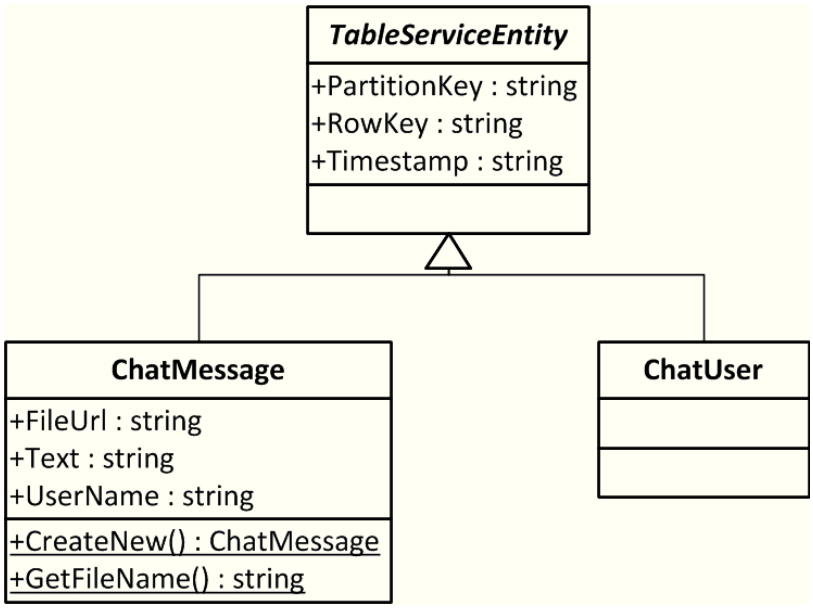
Seznam zpráv je ukládán v tabulce služby *Table Service* jako kolekce entit typu ChatMessage (obrázek 19). Eviduje se jméno autora (UserName), text zprávy a URL přiloženého souboru (FileUrl). Nejzajímavější je na této entitě způsob generování klíče RowKey (zdrojový kód 9). Hodnota klíče je s každou novou zprávou větší, přičemž reflektuje přesný čas vytvoření a zachovává si stejný počet číslic. Důvod tohoto způsobu generování klíče je fakt, že služba *Table Service* ukládá entity v pořadí určeném právě klíčem RowKey. Našim cílem je při dotazu na entity typu ChatMessage získat setříděnou kolekci seřazenou od nejnovější po nejstarší. Dále je zde zásadní možnost - dotázat se na kolekci entity novějších než stanovený čas (zdrojový kód 10).

```
ChatMessage chatMessage = new ChatMessage();
chatMessage.RowKey = (DateTime.MaxValue.Ticks − DateTime.Now.Ticks).ToString();
```

Výpis 9: Generování klíče RowKey entity ChatMessage

```
public IEnumerable<ChatMessage> FindAll(string lastSyncTime)
{
    //lastSyncTime = RowKey nejnovější instance třídy ChatMessage

    return _context.Messages
                .Where(m => m.PartitionKey == ChatMessage.PartitionKeyValue &&
                        m.RowKey.CompareTo(lastSyncTime) < 0).ToList();
}
```

Výpis 10: Získání kolekce entit typu ChatMessage



Obrázek 19: Entity aplikace *CloudChat*

3.5.2 Soubory přiložené ke zprávám

Soubory jsou ukládány jako veřejné bloby služby *Blob Storage* (kapitola 3.2.1). Výsledné URL blobu je přiřazeno příslušné entitě ChatMessage jako hodnota vlastnosti FileUrl.

Pokud uživatel zadá komprimaci souboru, vloží se do fronty služby *Queue Service* (kapitola 3.2.3) záznam (třída WokerJob) obsahující uživatelské jméno, text zprávy a URL originálního (nezkomprimovaného) souboru. Tento záznam je vyzvednut pracovní rolí a následně zpracován.

Pracovní role zkomprimuje originální soubor a výsledek uloží jako nový soubor. Následně vytvoří zprávu (třída ChatMessage) ukazující na zkomprimovaný soubor. Nakonec originální soubor smaže.

3.5.3 Škálovatelnost

Počet instancí webové a pracovní role není omezen. Uživatel může být při každém HTTP požadavku obsluhován jinou instancí webové role, protože stavové informace (v tomto případě pouze uživatelské jméno) je ukládáno na straně klienta ve formě *cookie*.

3.5.4 Použité technologie

Aplikace *CloudChat* je vytvořena nad platformou .NET Framework verze 3.5. Webová role je realizována pomocí technologie ASP.NET MVC 2 a jQuery 1.4.1. Ke komprimaci souborů je použita volně šiřitelná komponenta *DotNetZip Library*. Tato komponenta bylo

upřednostněna před třídou GZipStream vestavěnou v platformě .NET Framework z důvodu její mnohem větší efektivity.

4 Velmi náročné výpočty pomocí HPC a Windows Azure

Velmi náročnými výpočty (dále jen VNV) zde rozumíme výpočty, které jsou pro standardní počítače současnosti příliš náročné, tj. jejich výpočet by trval velmi dlouho.

High performance computing (HPC) se zabývá řešením VNV pomocí výkonných počítačů (např. ve formě clusteru, gridu apod.). Cílem této kapitoly je porovnat přístup řešení VNV pomocí HPC strojů a Windows Azure a zamyslet se nad jejich spoluprací.

Poznámky úvodem:

- V současné době se řešení VNV posunulo z oblasti primárně hrubého výkonu do oblasti paralelizace a distribuce. Z toho důvodu se text zaměřuje právě na hlediska paralelizace a distribuce.
- Z určitého úhlu pohledu by se dalo pojmem HPC označit i řešení VNV v prostředí Windows Azure. V následujícím textu se proto pojmem HPC myslí velmi výkonný počítač (cluster, grid) vně Windows Azure.

4.1 Paralelizace

Cílem paralelizace je využití vícero výpočetních jednotek k rychlejšímu řešení VNV. Zde se patří podotknout, že nejnáročnější fází paralelizace řešení nějaké VNV je samotná dekompozice řešení tak, aby se tížené paralelizace dalo opravdu efektivně využít. To ovšem přesahuje rámce tohoto textu a proto se tímto zde nebudeme příliš zabývat.

4.1.1 Úroveň vláken

Paralelizace pomocí vláken je z pohledu programátora tou nejnižší úrovní paralelizace [8]. Hned na začátku je třeba zdůraznit, že tato úroveň paralelizace je na HPC strojích i ve Windows Azure samozřejmě totožná. Na obou stranách je možno využít mnoho nástrojů pro práci s vlákny:

- API operačního systému (např. Win32 API).
- V případě programovacích jazyků C/C++ je velice populární asistovaná paralelizace pomocí nástroje OpenMP. Zdrojový kód se doplní informacemi o paralelizaci jednotlivých bloků pomocí direktivy `#pragma`. Před kompilací se spustí preprocesor OpenMP, který dle direktiv doplní příslušný kód pro práci s vlákny v daném operačním systému.
- Pro prostředí .NET Framework (od verze 4) je k dispozici knihovna *Task Parallel Library* (TPL). Jde o sadu tříd zaobalující standardní mechanismy pro práci s vlákny (např. třídy `Thread`, `ThreadPool`). Kromě vyšší úrovně abstrakce práci s vlákny přináší hlavně jejich dynamickou správu - vytváření a přidělování vláken dle běhového prostředí (počtu procesorů a jader).

[8] Existuje i nižší úroveň paralelizace - paralelní zpracování instrukcí v CPU.

Paralelizace pomocí vláken je v případě VNV využívána zejména jako sekundární způsob paralelizace. V drtivé většině případů hraje primární roli paralelizace na úrovni úloh.

4.1.2 Úroveň úloh

Úlohami jsou v případě HPC stroje myšleny jednotlivé procesy a v případě Windows Azure instance pracovních rolí [9]. Princip je v obou případech stejný - úlohy jsou spuštěny souběžně za účelem urychlení výpočtu výsledku, který získají vzájemnou komunikací.

Struktura úloh závisí na dekompozici paralelizovaného problému [4]:

Datová dekompozice – Vstupní data VNV jsou (pokud možno) rovnoměrně rozdělena na více částí, které jsou následně zpracovány instancemi určité úlohy.

Funkcionální dekompozice – Jednotlivým úlohám je přidělena určitá funkcionální role. Vděčným příkladem je simulace letadla - jedna úloha simuluje pohonné jednotky, další křídla atd.

Hybridní dekompozice – Typickým příkladem je model manažer/dělníci, kdy manažer přiděluje dělníkům (instancím nějaké pracovní úlohy) práci, tj. provádí datovou dekompozici. Z jiného pohledu jde o funkcionální dekompozici, protože manažer provádí jinou práci než dělníci.

4.2 Komunikace

U VNV je obecná snaha minimalizovat množství komunikace na minimum, protože se čekáním na I/O subsystémy ztrácí mnoho výpočetního času.

Řešení VNV můžeme dle komunikace v zásadě rozdělit do dvou skupin:

Komunikačně intenzivní – Typické pro sadu fyzicky blízkých souběžných úloh. Nejpoužívanější komunikační subsystémy jsou založeny na konceptu předávání zpráv.

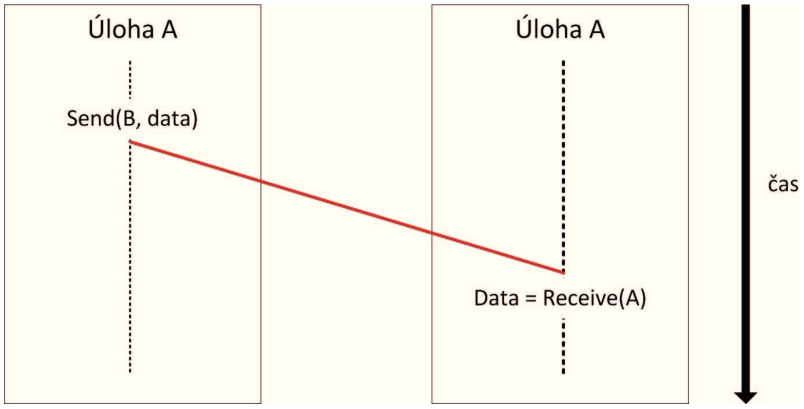
Webové služby – Jednotlivé úlohy jsou od sebe jak fyzicky, tak i koncepčně vzdálené. Běžné řešení u dlouho trvajících samostatných úloh.

4.2.1 Předávání zpráv

Model předávání zpráv (obrázek 20) je možno charakterizovat několika vlastnostmi [4]:

- Každá úloha má jedinečný identifikátor, díky němuž lze zprávy adresovat.
- Zprávy se skládají z obálky (zahrnující příjemce, odesílatele a další metadata) a z těla zprávy nesoucí data.

[9] Uvažovat zde webové role nemá význam.



Obrázek 20: Komunikace pomocí předávání zpráv

- Elementárními operacemi je *Send* (odeslání zprávy adresátovi) a *Receive* (příjem zprávy od odesílatele). Jde tedy o komunikaci bod-bod. Mimo to se hojně využívá i tzv. kolektivní komunikace, kdy se na určité operaci podílí všechny úlohy najednou.
- Existují operace určené pro synchronizaci úloh.

Nejznámějším standardem tohoto modelu je *Message Passing Interface* (MPI). Ve světě HPC existuje několik známých implementací tohoto standardu:

MPICH – Volně šiřitelná a přenositelná implementace.

OpenMPI – Open source implementace.

Intel MPI – Komerční implementace od společnosti Intel.

MPI.NET – Open source implementace na platformě .NET Framework 1.0.

Ve Windows Azure žádná implementace MPI standardu neexistuje. To ovšem neznamená, že myšlenky komunikace předávání zpráv nelze realizovat.

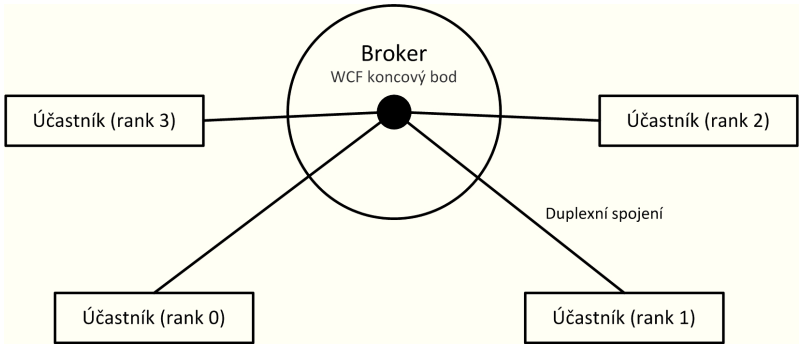
4.2.2 Příklad realizace modelu předávání zpráv ve Windows Azure

Princip předávání zpráv popsaný v předchozí části lze bez větších potíží implementovat i pomocí webových služeb. Jako příklad zde uvedu ukázkový systém *Message Passing over WCF*[10](obrázek 21).

Systém sestává ze speciální broker služby a libovolného počtu klientů (tj. účastníků). Broker má na starosti evidenci informací o každém připojeném účastníku, udržování duplexního spojení s každým účastníkem, synchronizaci účastníků a předávání zpráv mezi účastníky.

Každý účastník po připojení obdrží jedinečný identifikátor (rank), což je postupně inkrementovaná celočíselná hodnota od nuly.

[10]Tento systém je pouze demonstrativní pro účely této práce, rozhodně se nejedná o kompletní a vyladěný produkt. Zdrojové kódy naleznete na přiloženém CD.



Obrázek 21: Princpt systému *Message Passing over WCF*

Typický scénář komunikace vypadá následovně:

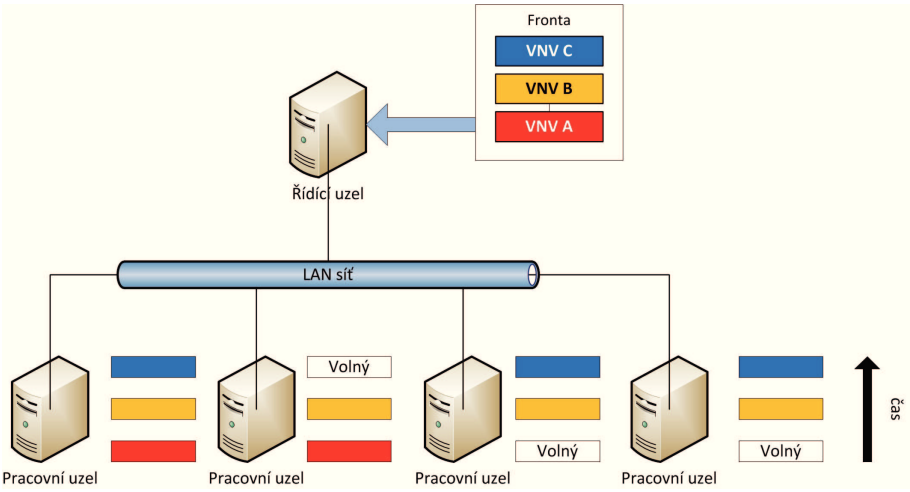
1. Broker je inicializován na zadané adrese a se zadaným počtem účastníků.
2. Každý účastník odešle na adresu brokeru žádost o připojení. Broker posečká, dokud nedostane počet žádostí odpovídající zadanému počtu účastníků. Jakmile se tak stane, každému účastníku zašle jeho rank a tím potvrdí úspěšné připojení.
3. Uvažujme, že účastník 0 chce zaslat zprávu účastníku 1. Účastník 0 tedy zašle zprávu brokeru (struktura zpráv je uvedena dále) a ten ji nasměruje do duplexního spojení asociované s rankem 1. Následně účastník 1 obdrží zprávu od brokeru.
4. Uvažujme, že se všichni účastníci chtějí v určitém bodu algoritmu synchronizovat. Jakmile se jednotlivý účastníci dostanou na místo synchronizace, odešlou zprávu brokeru. Ten se u každého účastníka poznamenává, zdali je synchronizován. Jakmile jsou synchronizováni všichni, broker synchronizaci všech účastníkům potvrdí zasláním notifikující zprávy.

Jak plyne i z názvu, systém je postaven na technologii WCF. Přesněji řečeno, ke komunikaci je využíván binding *netTcpBinding*, což zahrnuje duplexní a trvalé TCP spojení.

Zpráva se skládá z hlavičky obsahující ranky odesílatele a příjemce. Navíc obsahuje tzv. tag, což je libovolná uživatelsky definovaná číselná hodnota sloužící typicky k rozeznání typu zprávy. Tělo zprávy může obsahovat jakýkoli objekt serializovatelný do řetězce pomocí třídy DataContractSerializer.

Systém je možno ve Windows Azure nasadit následovně:

1. Vytvoří se pracovní role s jedinou instancí, která bude hostovat broker.
2. Pro každý typ úlohy se vytvoří pracovní role, což bude účastník.
3. Před spuštěním konkrétního VNV musíme znát celkový počet účastníků. Ten se nastaví v brokeru a nakonfiguruje se příslušný počet instancí na každé pracovní roli úloh.



Obrázek 22: Distribuce VNV na HPC clusteru

Je jistě patrné, že koncept předávání zpráv je přirozenější na HPC strojích. Ve Windows Azure se lépe uplatňují webové služby.

4.2.3 Webové služby

Jednotlivé instance úloh hrají roli černých skříněk, do kterých se vloží vstupní data, a počká se na výstup. Během zpracování požadavku se nepředpokládá žádná velká interakce s okolním světem.

Jde tedy a klasický model požadavek/odpověď, typicky pomocí SOAP zpráv na HTTP, TCP či MSMQ transportu.

4.3 Distribuce

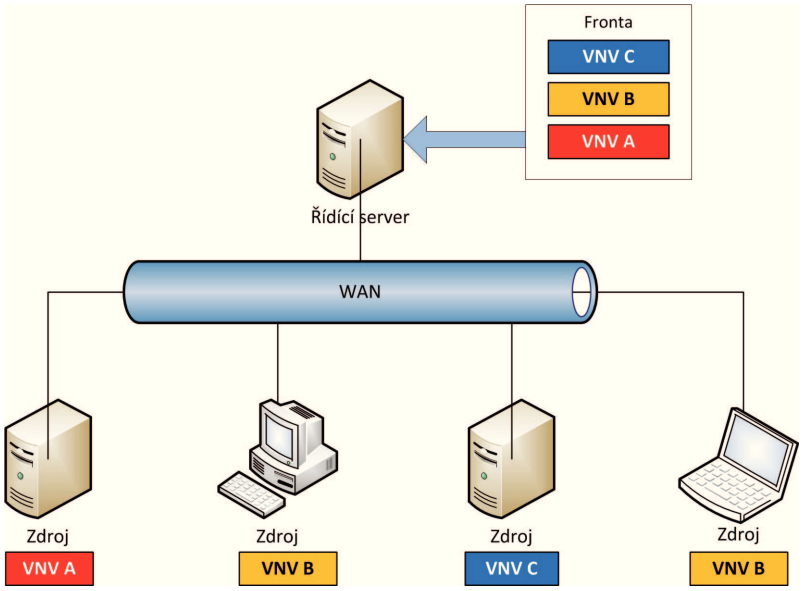
4.3.1 HPC

U HPC strojů se v praxi setkáváme se dvěma topologiemi: cluster a grid.

Jako *cluster* označujeme pevně spjatou soustavu počítačů (neboli uzlů), která se zvenčí tváří jako jeden stroj. Uzly jsou obvykle propojeny velmi rychlou LAN sítí a fyzicky se nacházejí velmi blízko. Soustavě pracovních uzlů často „velí“ řídící uzel. Pro cluster je typické, že všechny uzly jsou přibližně totožné (hardware, operační systém). Obrázek 22 ukazuje typickou distribuci VNV po clusteru.

Jako *grid* se označuje volně spjatá soustava počítačů (zdrojů) z různých administrativních domén za účelem řešení společného problému. Jednotlivé zdroje mohou být od sebe hodně vzdálené, takže komunikace probíhá i na úrovni WAN. Na rozdíl od clusteru pracuje i v heterogenním prostředí (tzn. různý hardware, operační systémy). Obrázek 23 zachycuje některé aspekty distribuce VNV v gridu[11]. VNV jsou distribuovány buď ve

[11] Ve své podstatě grid může rozdělovat pracovní zátěž stejně jako cluster, ale dále popisovaný způsob je pro grid typičtější.



Obrázek 23: Distribuce VNV na HPC gridu

větších „kusech“ nebo rovnou celé, protože přenášení dat mezi zdroji trvá mnohem déle než v případě clusteru. Pro grid je navíc typické, že se mezi jeho zdroje zapojují i méně výkonné počítače, viz. projekt *SETI@home*.

4.3.2 Windows Azure

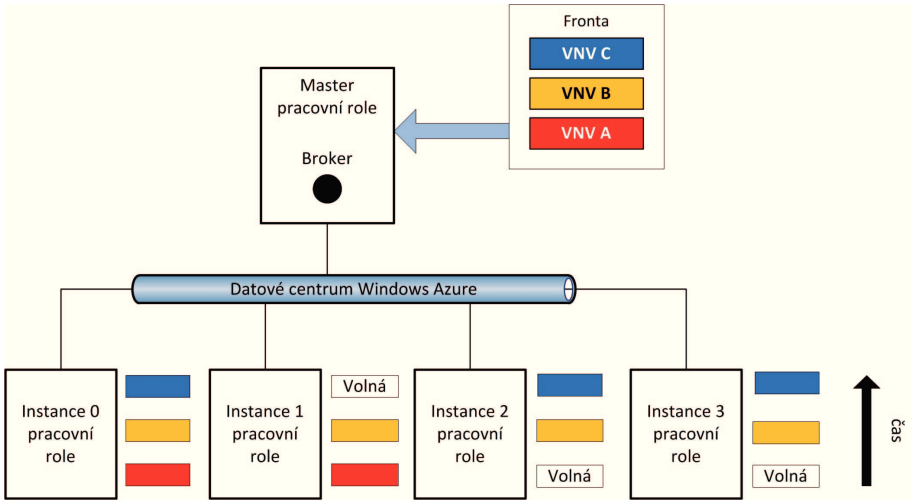
Pokračujme v myšlence předávání zpráv pomocí webových služeb (kapitola 4.2.2).

Z obrázku 24 je patrná podobnost s HPC clusterem. Ve skutečnosti je mezi tímto řešením a clusterem rozdíl v rychlosti komunikace. Nelze totiž očekávat, že by byla přenosová rychlost a latence sítě datového centra stejně velká jako LAN clusteru, viz. tabulka 4 demonstrující naměřené[12] přenosové rychlosti mezi webovou a pracovní rolí v rámci datových center Windows Azure.

Objem dat	Čas k přenesení (ms)	Přenosová rychlost (MB/s)
500 KB	92	5,4
3 MB	218	13,8
5 MB	330	15,2
10 MB	489	20,5
15 MB	579	25,9

Tabulka 4: Naměřené přenosové rychlosti mezi rolemi v rámci Windows Azure

[12] Zdrojové kódy testovací aplikace naleznete na přiloženém CD.



Obrázek 24: Distribuce VNV ve Windows Azure

Testovací data byla ve formě náhodných polí bytů a přenášení bylo realizováno technologií WCF (TCP).

4.4 Správa a plánování

Stroje řešící VNV mají vysoké provozní náklady. Proto se klade velký důraz na co nejefektivnější využití prostředků.

U HPC se často využívá určitá forma fronty VNV. K nim jsou přiřazeny metadata obsahující informace o požadovaných výpočetních prostředcích. Fronta je spravována službou obvykle v řídícím uzlu, jenž mimo jiné vybírá VNV z fronty, rezervuje požadované zdroje a spustí výpočet (obrázek 22). Díky tomuto přístupu má administrátor v každém okamžiku přesné informace o aktuálním i budoucím výpočetních využití zdrojů. Samozřejmostí je i nějaká forma inteligentního plánování.

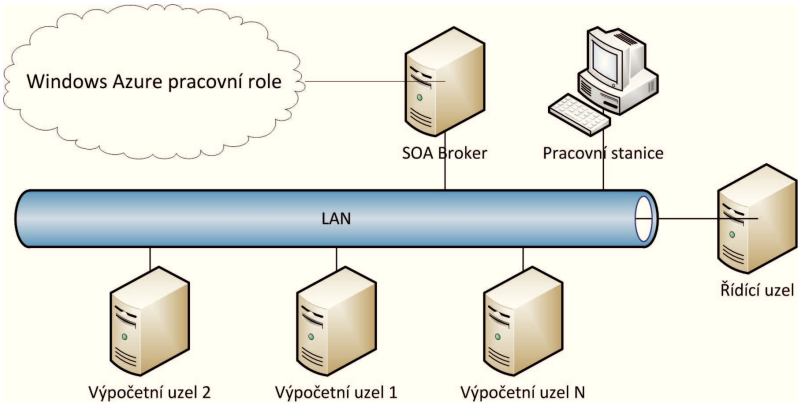
Ve Windows Azure je situace o poznání jednodušší. Neexistuje zde vestavěný způsob plánování využití jednotlivých služeb. Existuje ovšem poměrně bohaté API rozhraní pro jejich správu. Je tedy čistě na administrátorovi, aby spouštěl instance pracovních rolí či spravoval datová úložiště ručně, nebo plánovaně za asistence skriptů.

4.5 Spolupráce Windows HPC a Windows Azure

Následující text se bude zabývat spoluprací clusteru řízeným systémem Microsoft Windows HPC Server 2008 R2 a pracovními rolemi Windows Azure.

Řešení VNV výhradně ve Windows Azure není technicky ani ekonomicky výhodné:

- Některé VNV pracují s citlivými daty, které musí mít organizace stále pod přísnou kontrolou a proto je nemožné takoví data ukládat nebo zpracovávat ve Windows Azure.



Obrázek 25: Windows Azure rozšiřující Windows HPC cluster

- Mnoho VNV využívají softwarové komponenty třetích stran, které ve většině případů nejsou připraveny na prostředí Windows Azure.
- VNV často pracují s velkými objemy dat. Jejich přesun do úložišť Windows Azure by mohl být problematický.

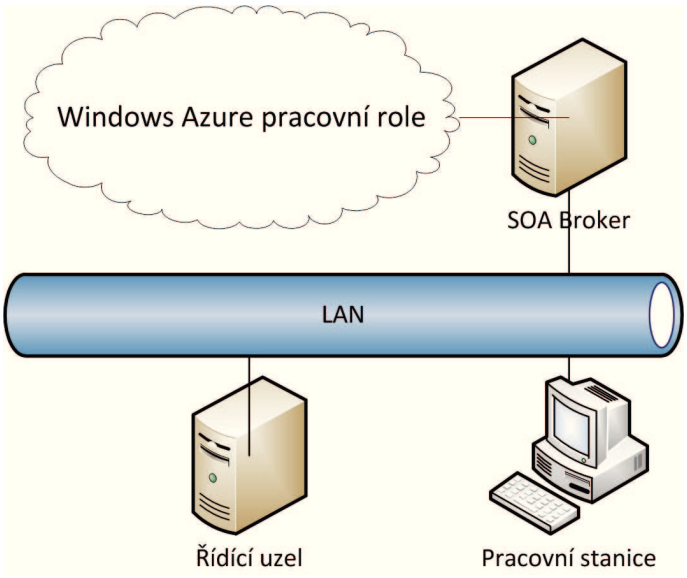
Vhodným řešením je spolupráce Windows HPC a Windows Azure. V zásadě existují dva přístupy:

- Využít pracovní role ve Windows Azure jako pomocné uzly ve Windows HPC.
- Veškeré výpočetní uzly HPC nahradit pracovními rolemi ve Windows Azure.

Obrázek 25 ilustruje Windows HPC cluster rozšířený o pracovní role ve Windows Azure. Veškerá infrastruktura Windows HPC je na svém místě. Jediným rozdílem je, že do SOA brokeru jsou kromě interních výpočetních uzlů zapojeny i pracovní role ve Windows Azure. Pracovní role ve Windows Azure nejsou vytvářeny dynamicky při přidání nové pracovní položky do Windows HPC. Administrátor je musí explicitně alokovat a po skončení výpočtu uvolnit[13].

Obrázek 26 ukazuje Windows HPC cluster využívající výhradně pracovní role ve Windows Azure. V tomto případě jsou veškeré výpočetní kapacity externalizovány do Windows Azure. HPC zde hraje roli koordinátora.

[13]Tuto proceduru lze samozřejmě automatizovat pomocí skriptů.



Obrázek 26: Windows HPC cluster využívající výhradně Windows Azure

5 Závěr

Z historického pohledu je cloud computing relativně mladá disciplína, ale poslední léta ukázala, že jde o správnou cestu. Výpočetní výkon se nezadržitelně zvyšuje a díky cloud computingu i efektivita. Cílem je, aby každá organizace disponovala jenom takovým výpočetním výkonem, který dokáže využít. Ušetří tím nejenom náklady, ale i životní prostředí.

Platforma Windows Azure má za sebou pouze několik let a to je bohužel znát. Architektura i technologie jsou sice na špičkové úrovni, ale při vývoji aplikací jsem narazil na několik „nedodělků“. Patří mezi ně zejména nemožnost nastavení limitu spotřebovávaných výpočetních prostředků, několik měsíců neřešená chyba při hostování HTTP WCF služby v pracovní roli a velmi omezené možnosti služby *Table Service*. Na obhajobu je třeba podotknout, že Windows Azure je v překotném vývoji, takže uvedené výhrady již nemusí být za pár měsíců skutečností.

Osobně jsem měl před touto prací jen malé zkušenosti s Windows Azure. Při vytváří této práce jsem si ujasnil význam cloud computingu v širším kontextu, prohloubil znalosti Windows Azure a seznámil jsem se i s ostatními platformami cloud computingu.

6 Literatura

[1] BAR, Jeff. *Host Your Web Site in the Cloud: Amazon Web Services Made Easy*. Seattle : Amazon Web Services, 2010. 300 s. ISBN 978-0-9805768-3-2. [kniha]

[2] BERTOCCI, Vittorio. *Programing Windows Identity Foundation*. Redmond : Microsoft Press, 2010. 272 s. ISBN 978-0-7356-2718-5. [kniha]

[3] CHRIS, Hay; PRINCE, Brian H. *Azure in Action*. Stamford : Manning Publications, 2010. 425 s. ISBN 978-1935182481. [kniha]

[4] FOSTER, Ian. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Boston : Addison Wesley, 1995. 430 s. ISBN 978-0201575941. [kniha]

[5] HURWITZ, Judith; BLOOR, Robin; KAUFMAN, Marcia. *Cloud Computing For Dummies*. Indianapolis : Wiley Publishing, 2010. 310 s. ISBN 978-0-470-48470-8. [kniha]

[6] LOWY, Juval. Working With The .NET Service Bus. *MSDN Magazine* [online]. Duben 2009, č. 110, [cit. 2011-02-08]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/magazine/dd569756.aspx>. [e-článek]

[7] LOWY, Juval. Routers in the Service Bus. *MSDN Magazine* [online]. Říjen 2009, č. 116, [cit. 2011-02-08]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/magazine/ee335696.aspx>. [e-článek]

[8] REDKAR, Tejaswi. *Windows Azure Platform*. New York : Apress, 2009. 609 s. ISBN 978-1430224792. [kniha]

[9] RITTINGHOUSE, John W.; RANSOME, James F. *Cloud Computing: Implementation, Management*, and Security. Boca Raton : CRC Press, 2010. 301 s. ISBN 978-1439806807. [kniha]

[10] SARDENSON, Dan. *Programming Google App Engine*. Sebastopol : O'Reilly Media, 2010. 367 s. ISBN 978-0-596-52272-8. [kniha]

[11] VELTE, Anthony T.; VELTE, Toby J.; ELSENPETER, Robert. *Cloud Computing: A Pratical Approach*. New York : The McGraw-Hill Companies, 2010. 334 s. ISBN 978-0-07-162695-8. [kniha]